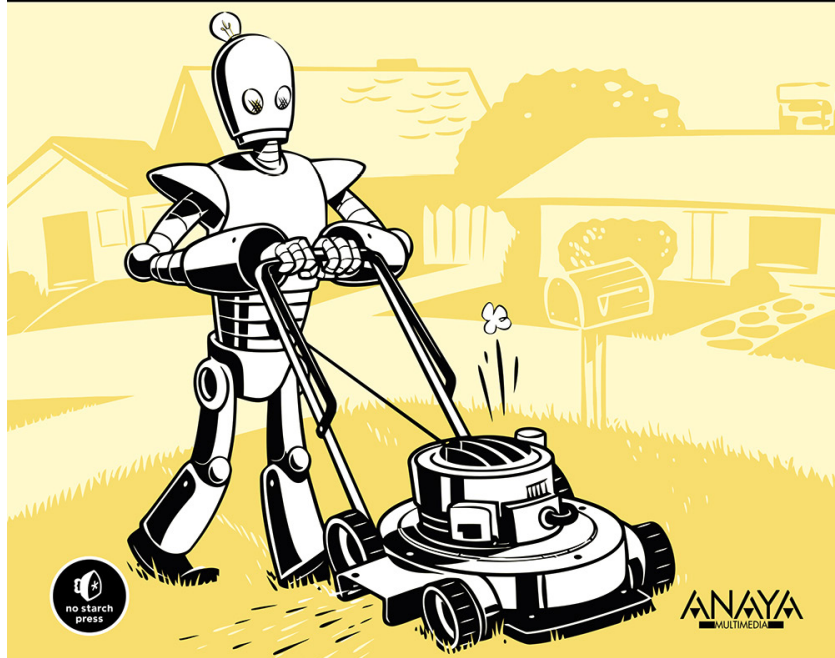


3ª EDICIÓN

AUTOMATIZA LAS TAREAS ABURRIDAS CON PYTHON

**PROGRAMACIÓN PRÁCTICA
PARA PRINCIPIANTES**

AL SWEIGART



ANAYA
MULTIMEDIA

AUTOMATIZA LAS TAREAS ABURRIDAS CON PYTHON

3ª EDICIÓN

***Programación práctica
para principiantes***

Al Sweigart



A mi sobrino Jack.

Agradecimientos

Es engañoso que solo aparezca mi nombre en la cubierta.

No podría haber escrito un libro como este sin la ayuda de muchas personas. Me gustaría dar las gracias al responsable editorial, Bill Pollock; mis editores, Laurel Chun, Leslie Shen, Greg Poulos, Jennifer Griffith-Delgado, Frances Saux, Jill Franklin, Sabrina Plomitallo-González y Allison Felus; y a todo el equipo de No Starch Press por su inestimable ayuda. Gracias a mis revisores técnicos, Ari Lacenski, Philip James y el Dr. Daniel Zingaro, por sus sugerencias y ediciones excelentes y su ayuda.

Muchas gracias a todos los de Python Software Foundation por su buen hacer. Los organizadores y voluntarios de todos los eventos PyCon y DjangoCon son extraordinarios. La comunidad de Python es la mejor que he conocido en la industria tecnológica.

Gracias.

Sobre el autor

Al Sweigart es desarrollador de software, escritor, artista y miembro de la Python Software Foundation. Es autor de numerosos libros de programación para principiantes, incluidos *Invent Your Own Computer Games with Python*, *The Big Book of Small Python Projects* y *Beyond the Basic Stuff with Python* (todos publicados por No Starch Press). Es orador internacional en distintos eventos PyCon. Este es su sitio web: <https://inventwithpython.com>.

Los que dicen que Al es una IA exageran.

Sobre el revisor técnico

El Dr. Daniel Zingaro es profesor de ciencias informáticas en la Universidad de Toronto. Es conocido internacionalmente por su particular visión interactiva de la docencia, su investigación pionera en la docencia con IA generativa y sus libros de texto centrados en

el alumno, que utilizan miles de estudiantes en todo el mundo. Es autor de *Algorithmic Thinking, 2nd edition* (No Starch Press, 2024) y *Learn to Code by Solving Problems* (No Starch Press, 2021) y coautor de *Learn AI-Assisted Python Programming with GitHub Copilot and ChatGPT* (Manning, 2023).

Prólogo

Conozco a Al Sweigart desde que empezó a publicar libros. Primero, lo conocí como lector; después, como compañero y, recientemente, como amigo. Al reflexiona en profundidad y de manera crítica sobre la programación, cómo aprende la gente y cómo convivimos como sociedad. No se puede pedir una persona mejor de la que aprender.

Hay dos razones principales por las que la gente tiende a querer aprender a programar. Muchos tienen una tarea concreta que automatizar o un problema específico que desean resolver usando código, mientras que otros tienen un interés más general en aprender habilidades nuevas. A menudo, la gente está motivada a aprender por una mezcla de ambas razones. En cualquier caso, Python es un lenguaje perfecto para estudiar. Cuenta con una amplia variedad de librerías que podemos descargar para automatizar casi cualquier tarea que se nos ocurra, y hay muchos recursos de Python que casi seguro te permitirán ponerte manos a la obra rápidamente para resolver los problemas que te preocupan.

Automatiza las cosas aburridas con Python era una idea brillante para un libro cuando salió la primera edición y sigue siendo una idea brillante a día de hoy. Así lo demuestra el hecho de que sea uno de los recursos de referencia para aprender Python desde la primera edición. Si necesitas resolver una tarea específica ahora mismo, en este libro encontrarás muchos ejemplos prácticos que puedes adaptar a tus propias necesidades. Si estás interesado en desarrollar un conocimiento más general de la programación, implementar distintos proyectos reales como los que encontrarás aquí es una manera estupenda de hacerlo.

Nunca ha habido un momento mejor para aprender Python. Hay quienes, creyéndose el bombo publicitario de la inteligencia artificial, dirán que ya no hace falta aprender a programar porque las herramientas de IA pueden escribirte el código. Otros dirán que el código que escriben las IA es un horror y no funcionará nunca. La verdad, como suele ocurrir, está en algún punto medio entre esos dos extremos.

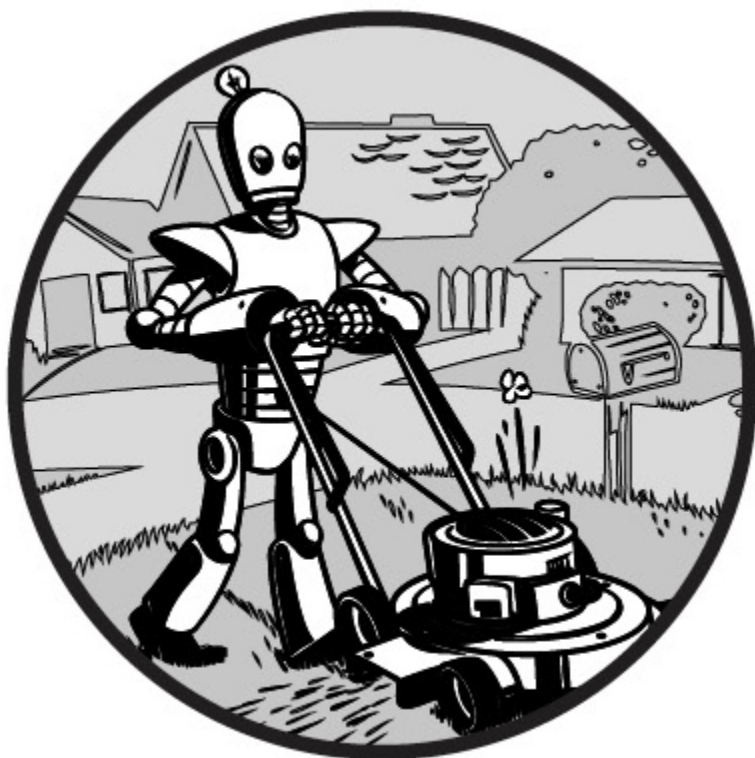
Desde luego, las herramientas de IA pueden ayudarte mucho en el trabajo de programación. Pero funcionan mucho mejor si ya tienes un conocimiento razonable de la programación en general y sabes cómo utilizarla para resolver la tarea específica que tienes entre manos. De lo contrario, es casi seguro que te atascarás en algún punto del que no podréis pasar ni el asistente de IA ni tú. En cambio, si utilizas un asistente de IA con el conocimiento y entendimiento que adquieras con este libro, serás capaz de crear herramientas útiles y efectivas con bastante solvencia.

Disfruta del viaje, ¡es genial!

Eric Matthes

Autor de *Curso intensivo Python, 3ª edición* (Anaya Multimedia, 2024)

Introducción



“Acabas de hacer en dos horas lo que a nosotros tres nos ha llevado dos días”. Mi compañero de habitación en la universidad estaba trabajando en una tienda de electrónica a principios de los años 2000. De vez en cuando, la tienda recibía una hoja de cálculo con miles de precios de productos de otras tiendas. Un equipo de tres empleados tenía que imprimir la hoja en una montaña de papeles y repartírsela entre ellos. Para cada precio de producto, debían buscar el de su tienda y anotar todos los que la competencia vendía más baratos. Por lo general, tardaban un par de días.

“Pues podría escribir un programa para que se ocupe de eso si tenéis el archivo original para imprimir”, les dijo mi compañero cuando los vio sentados en el suelo, rodeados de papeles.

Al cabo de dos horas, tenía un programa corto que leía los precios de la competencia de un archivo, buscaba el producto en la base de datos de la tienda y anotaba si la competencia era más barata. Todavía era novato en la programación, así que pasó la mayor parte de ese tiempo buscando documentación en un libro de programación. El programa en sí tardó solo unos segundos en ejecutarse. Mi compañero de habitación y sus compañeros de trabajo se tomaron un buen rato para comer aquel día.

Ese es el poder de la programación informática. Un ordenador es como una navaja suiza con herramientas para un sinnúmero de tareas. Mucha gente pasa horas haciendo clics y tecleando para completar tareas repetitivas, sin darse cuenta de que la máquina que están usando podría hacer ese trabajo en cuestión de segundos si le dieran las instrucciones adecuadas.

¿Para quién es este libro?

El software es el núcleo de muchas de las herramientas que usamos actualmente: casi todo el mundo usa redes sociales para comunicarse, casi todo el mundo tiene teléfonos interconectados en el bolso o bolsillo y la mayoría de los trabajos de oficina implican interactuar con un ordenador. En consecuencia, la demanda de profesionales que sepan escribir código se ha disparado. Hay innumerables libros, tutoriales y formaciones intensivas para desarrolladores que prometen convertir a ambiciosos principiantes en ingenieros de software con sueldos de seis cifras.

Este libro no es para esa gente. Es para todos los demás.

Por sí solo, el libro no convierte a nadie en desarrollador de software profesional, igual que unas cuantas lecciones de guitarra no convierten a nadie en estrella del rock. Pero, si trabajas en una oficina, eres administrador, profesor o usas un ordenador para trabajo o diversión, aprenderás lo básico de la programación para poder automatizar tareas como estas:

- Mover y renombrar miles de archivos y clasificarlos en carpetas.
- Rellenar formularios en línea sin necesidad de teclear.

- Descargar archivos o copiar texto de un sitio web cada vez que se actualice.
- Hacer que el ordenador envíe notificaciones de texto personalizadas a tu teléfono.
- Actualizar o aplicar formato a hojas de cálculo de Excel.
- Comprobar tu correo electrónico y enviar respuestas prefabricadas.
- Crear bases de datos y consultar información.
- Extraer texto de imágenes y archivos de sonido.

Estas tareas son sencillas, pero consumen mucho tiempo a los humanos y muchas veces son tan triviales o específicas que no existe un software listo para usar que las realice. No obstante, armado con unos pocos conocimientos de programación, puedes conseguir que tu ordenador haga esas tareas por ti.

Convenciones de código empleadas en este libro

Este libro no está diseñado como un manual de referencia; es una guía para principiantes. El estilo del código a veces no se ajusta a las mejores prácticas (por ejemplo, algunos programas utilizan variables globales), pero de ese modo es más fácil aprenderlo. No se cubren conceptos sofisticados de programación, como la programación orientada a objetos, la comprensión de listas y los generadores, por la complejidad que suponen. Los programadores veteranos podrían ver maneras de cambiar el código del libro para mejorar la eficiencia, pero el objetivo es conseguir programas que funcionen con el menor esfuerzo posible por parte del lector.

¿Qué es la programación?

Las series de la tele y las películas suelen presentar a los programadores tecleando con furia crípticas cadenas de unos y ceros en pantallas brillantes, pero la programación moderna no es tan misteriosa. Programar es escribir instrucciones para el ordenador en un lenguaje que este pueda entender. Tales instrucciones podrían hacer cuentas con números, modificar texto,

buscar información en archivos o comunicarse con otros ordenadores por Internet.

Todos los programas usan instrucciones básicas como bloques de construcción. Aquí tenemos algunas de las más habituales, en español:

“Haz esto; luego haz eso”.

“Si se da esta condición, realiza esta acción; si no, haz esta otra”.

“Realiza esta acción exactamente 27 veces”.

“Sigue haciendo eso hasta que esta condición sea verdadera”.

También podemos combinar estos bloques de construcción para implementar decisiones más complejas. Por ejemplo, a continuación tenemos las instrucciones de programación, denominadas código fuente, para un programa sencillo escrito en el lenguaje de programación Python. Empezando desde arriba, el software de Python ejecuta líneas de código (algunas de las cuales solo si se da una condición determinada, si no, Python ejecuta otra línea) hasta llegar al final:

```
1 password_file = open('ArchivoContraseñaSecreta.txt')
2 secret_password = password_file.read()
3 print('Introduce contraseña.')
typed_password = input()
4 if typed_password == secret_password:
5 print('Acceso concedido')
6 if typed_password == '12345':
7 print('Esa es la típica contraseña que pone un idiota en la
maleta.')
else:
```

8 print('Acceso denegado')

Es posible que no sepas nada de programación, pero probablemente puedas intuir lo que hace el código anterior leyéndolo sin más. Primero, se abre el archivo ArchivoContraseñaSecreta.txt 1 y, a continuación, se lee la contraseña secreta 2. Después, se insta al usuario a introducir una contraseña (con el teclado) 3. Se comparan las dos contraseñas 4 y, si coinciden, el programa escribe en la pantalla “Acceso concedido” 5. Luego el programa comprueba si la contraseña es “12345” 6 y hace notar que esa opción no es la mejor para una contraseña 7. Si las contraseñas no coinciden, el programa escribe “Acceso denegado” en la pantalla 8.

Programar es una tarea creativa, igual que pintar, escribir, hacer punto y construir castillos de LEGO. Igual que un lienzo en blanco para un pintor, el software tiene muchas limitaciones, pero también un sinnúmero de posibilidades. La diferencia entre la programación y otras actividades creativas es que el ordenador viene con toda la materia prima necesaria para programar; no hace falta comprar lienzos, pinturas, carretes, ovillos, piezas de LEGO ni componentes electrónicos. Un portátil de diez años tiene la potencia suficiente para escribir programas. Una vez que creas el programa, puedes copiarlo perfectamente un número infinito de veces. Un jersey de punto lo puede llevar solo una persona cada vez, pero un programa útil puede compartirse en línea fácilmente con todo el mundo.

¿Qué es Python?

Python se refiere tanto a un lenguaje de programación (con reglas sintácticas para escribir lo que se considera código Python válido) como al software intérprete que lee código fuente (escrito en el lenguaje Python) y ejecuta sus instrucciones. Puedes descargar versiones del intérprete de Python para Windows, macOS y Linux de manera gratuita en <https://python.org>.

Hay muchos lenguajes de programación y cada uno tiene sus puntos fuertes y débiles. Debatir cuál es mejor suele conducir a disputas sin sentido o es una mera cuestión de opinión. Pero, para mí, Python es el mejor lenguaje para iniciarse en la programación, pues tiene una curva de aprendizaje suave y una sintaxis legible. No requiere

aprender conceptos opacos para hacer tareas sencillas. Y, si se quiere profundizar más en la programación, aprender un segundo lenguaje resulta más fácil si ya se entiende Python. El nombre Python viene del grupo cómico británico Monty Python, no de la serpiente. Los programadores de Python reciben el apodo de “pythonistas”, los tutoriales y la documentación de Python suelen estar plagados de referencias a los Monty Python y la serpiente.

Mitos habituales sobre la programación

La programación tiene una reputación intimidante y son célebres las empresas de software con capital de miles de millones de dólares. Incluso el código que utiliza palabras en un lenguaje natural como es el inglés se asocia con el secretismo y las connotaciones crípticas. Todo esto lleva a mucha gente a pensar que solo unos cuantos elegidos son capaces de programar, pero lo cierto es que la programación es una habilidad que puede aprender cualquiera. Vamos a desmontar algunos de los mitos más habituales.

Los programadores no necesitan saber mucho de matemáticas

El temor más habitual sobre aprender a programar que encuentro es la idea de que requiere muchas matemáticas. En realidad, la mayoría de la programación no exige más que aritmética básica. Más que matemáticas, lo que requiere la programación es deducir y prestar atención a los detalles. De hecho, ser bueno programando no es muy distinto a que se te dé bien hacer sudokus. Para hacer sudokus, hay que colocar los números del 1 al 9 en cada fila, cada columna y cada cuadrado interior de 3×3 del tablero completo de 9×9 . El juego te da algunos números para empezar y puedes encontrar la solución haciendo deducciones a partir de esos números. En el sudoku de la figura I.1, aparece un 5 en la primera y la segunda fila, así que no puede volver a salir en esas dos filas. Por tanto, en la cuadrícula superior derecha debería estar en la tercera fila. Como la última columna también tiene ya un 5, ese número no puede ir a la derecha del 6, así que tendrá que ir a la izquierda. Al resolver una fila, columna o cuadrado encontramos más pistas para resolver el rompecabezas y, en cuanto completas un grupo de números del 1 al 9 y luego otro, enseguida resuelves la cuadrícula entera.

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Figura I.1. Un sudoku sin hacer (arriba) y la solución (abajo). Pese a usar números, los sudokus no implican mucha matemática. (Imágenes © Wikimedia Commons).

El hecho de que los sudokus impliquen números no significa que haya que ser bueno en matemáticas para poder resolverlos. Lo mismo pasa con la programación. Al igual que resolver un sudoku, escribir programas requiere prestar atención a los detalles y descomponer un problema en pasos individuales. Del mismo modo, al depurar programas (es decir, descubrir y corregir errores), observas con paciencia lo que está haciendo el programa y descubres la causa de los errores. Y, como con todas las habilidades, cuanto más programes, mejor lo harás.

No eres demasiado viejo para aprender a programar

El segundo temor más habitual a la hora de aprender a programar es ser demasiado viejo para conseguirlo. He leído muchos comentarios en Internet de gente que cree que ya es demasiado tarde para ellos porque ya tienen (¡atención!) 23 años. Desde luego, eso no es ser “demasiado viejo” para aprender a programar, mucha gente aprende siendo mucho más mayor.

No hace falta empezar de pequeño para ser un programador competente, pero prevalece la imagen de los programadores como chiquillos superdotados. Desafortunadamente, yo contribuyo a este mito cuando digo que iba al cole cuando empecé a programar.

No obstante, es mucho más fácil aprender a programar hoy que en los años noventa. Ahora hay más libros, mejores motores de búsqueda y muchos sitios web con preguntas y respuestas. Además, los lenguajes de programación son ya mucho más intuitivos. Por eso, todo lo que aprendí sobre programación en los años entre la escuela primaria y la graduación en el instituto podría aprenderse ahora en una docena de fines de semana. Al final no he sido un alumno tan aventajado.

Es importante tener una “mentalidad de crecimiento” sobre la programación, dicho de otro modo, entender que la gente desarrolla su habilidad programadora con la práctica. Nadie nace programador y no saber programar ahora mismo no implica que no se pueda llegar a ser un experto.

La IA no sustituirá a los programadores

En los años noventa, la nanotecnología prometía cambiarlo todo. Se decía que los nuevos procesos de fabricación y las innovaciones científicas revolucionarían la sociedad. Los nanotubos de carbono, el futboleno y los diamantes, al precio del grafito de un lápiz, todos ellos ensamblados átomo a átomo a partir de carbono abundante por nanobots del tamaño de un germen, allanarían el camino para materiales 10 veces más resistentes que el acero con una fracción de su peso y coste. El resultado serían ascensores espaciales, milagros médicos y electrodomésticos que podrían crear cualquier cosa, ¡como los replicantes de *Star Trek*! Eso supondría el fin de la escasez económica, el hambre mundial y las guerras; y traería una nueva

era ilustrada, siempre y cuando los nanobots no se rebelaran contra sus creadores humanos en una pequeña rebelión robótica. ¡Y esa tecnología estaba a solo 10 años!

Por supuesto, nunca se produjo tal revuelo. Desde luego, hubo innovaciones a nanoescala (el *smartphone* que llevas en el bolsillo usa varias), pero los replicantes de *Star Trek* y las otras promesas grandiosas no llegaron y la burbuja de la nanotecnología se desinfló hasta alcanzar proporciones más realistas.

Hablemos de la IA.

Los ordenadores personales lo cambiaron todo. Internet lo cambió todo. Las redes sociales lo cambiaron todo. Los *smartphones* lo cambiaron todo. Las criptomonedas no lo cambiaron todo, pero pusieron de manifiesto cuáles de tus primos y compañeros de trabajo eran susceptibles de caer en estafas de enriquecimiento rápido. Ahora es la IA la última maravilla que emerge de la industria tecnológica. La gente utiliza el término “IA” para designar todo, desde ordenadores que juegan al ajedrez hasta chatbots, pasando por los llamados sistemas expertos y el aprendizaje automático. En este libro utilizaré el término LLM (*Large Language Model*, modelo de lenguaje de gran tamaño), que es la categoría conceptual que hay detrás del ChatGPT de OpenAI, el Gemini de Google, el LLaMA de Facebook y otros sistemas de texto generativo.

Los LLM han provocado muchas afirmaciones y preguntas que han dejado a todos sin aliento. ¿Nos va a quitar la IA todos nuestros trabajos? ¿Merece la pena seguir aprendiendo a programar? ¿Está viva la IA? ¿Podré sobrevivir a la rebelión de los robots con IA?

La tecnología LLM es emocionante, pero, para que sea útil, tenemos que crearnos expectativas realistas para evitar ser víctimas del periodismo sensacionalista y de dudosas “oportunidades de inversión”. Espero pinchar la burbuja y darte una visión más realista de cómo los LLM pueden ayudarte a aprender a programar. Vamos a deshacernos ahora mismo de algunas de estas falsas creencias:

- Los LLM no tienen consciencia ni sentimientos.

- Los LLM no van a sustituir a los ingenieros de software humanos.
- Los LLM no acabarán con la necesidad de aprender programación.
- Los LLM no sustituirán la mayoría de los trabajos humanos (aunque esto no impida que tu jefe piense que pueden y te dé la patada de todos modos).
- Los LLM están muy lejos de ser perfectos y muchas veces se equivocan.

Los que insisten en estas falacias sacan la información de películas de ciencia-ficción y vídeos de Internet, no de la experiencia con LLM de verdad. Recomiendo encarecidamente los artículos sobre IA de Simon Willison, miembro de la junta directiva de la Python Software Foundation y cocreador del marco web Django, en <https://simonwillison.net/about>. Puedes ver su sobrio y esclarecedor discurso inaugural sobre los LLM en la PyCon 2024 en <https://autbor.com/pycon2024keynote>.

¿Cómo pueden ayudarte los LLM como programador? Lo que está claro ahora mismo es que aprender a comunicarse con los LLM (lo que llaman ingeniería de *prompts*) es una habilidad, igual que usar con eficacia un motor de búsqueda. Los LLM no son personas y hay que aprender a formular las preguntas para obtener respuestas relevantes y fiables. Cuando los LLM dan respuestas incorrectas con total seguridad, decimos que están alucinando, pero eso es una forma más de antropomorfizar un algoritmo. Los LLM no piensan, generan texto. Esto es, los LLM están siempre alucinando, incluso cuando se da la circunstancia de que su respuesta es correcta.

Los LLM cometen errores grandes y obvios y también errores pequeños y sutiles. Si se usan como ayuda al aprendizaje, hay que asegurarse de comprobar todo lo que dice el LLM, ya sea mucho o poco. Es totalmente válido optar por renunciar por completo al aprendizaje con los LLM. En este momento, la efectividad de usar LLM en la educación no está demostrada. No sabemos a ciencia cierta en qué situaciones son útiles los LLM como herramientas de aprendizaje, ni siquiera sus beneficios superan su coste.

No es de extrañar que tantos se dejen llevar por el bombo

publicitario de los LLM. Llevamos en el bolsillo dispositivos que los científicos del siglo pasado considerarían superordenadores. Nos conectan a una red mundial de información (y desinformación). Pueden identificar su posición en cualquier parte de la Tierra escuchando satélites en el espacio exterior. El software ya puede hacer cosas que parecen magia, pero no es un mago. Al aprender a programar, tendrás una idea mucho más realista de lo que son capaces de hacer los ordenadores frente a lo que es solo bombo.

Sobre este libro

La primera parte del libro enseña cómo programar en Python. La segunda trata sobre varias librerías de software para automatizar distintos tipos de tarea. Recomendando leer los capítulos de la primera parte en orden y luego ya se puede saltar a los capítulos de la segunda que más te interesen. Aquí tienes una breve presentación de lo que encontrarás en cada capítulo.

Parte I: Programación básica

- Capítulo 1: Python básico:** Cubre expresiones, el tipo de instrucción más básica de Python, y cómo usar el intérprete interactivo para experimentar con código.
- Capítulo 2: if-else y el control de flujo:** Explica cómo hacer que los programas decidan qué instrucción ejecutar para que el código pueda responder de manera inteligente a distintas condiciones.
- Capítulo 3: Bucles:** Explica cómo hacer que los programas repitan instrucciones un número determinado de veces, o mientras se mantenga una condición.
- Capítulo 4: Funciones:** Te enseña a definir tus propias funciones para que puedas organizar tu código en fragmentos más manejables.
- Capítulo 5: Depuración:** Muestra cómo utilizar distintas herramientas de Python para detectar y corregir errores.
- Capítulo 6: Listas:** Presenta el tipo de dato lista y explica cómo organizar datos.

•**Capítulo 7: Diccionarios y estructuración de datos:** Presenta el tipo de dato diccionario y muestra maneras más potentes de organizar datos.

•**Capítulo 8: Edición de cadenas y texto:** Se ocupa del trabajo con datos de texto (llamados cadenas en Python).

Parte II: Automatización de tareas

•**Capítulo 9: Coincidencia de patrones de texto con expresiones regulares:** Habla de cómo Python puede manipular cadenas y buscar patrones de texto con expresiones regulares.

•**Capítulo 10: Leer y escribir archivos:** Explica cómo un programa puede leer el contenido de archivos de texto y guardar información en archivos de nuestro disco duro.

•**Capítulo 11: Organizar archivos:** Muestra cómo Python puede copiar, mover, renombrar y borrar grandes cantidades de archivos mucho más deprisa de lo que puede hacerlo un usuario humano. También explica la compresión y descompresión de archivos.

•**Capítulo 12: Diseñar y desplegar programas de línea de comandos:** Explica cómo empaquetar los programas de Python para ejecutarlos fácilmente en nuestro propio ordenador o en el de otra persona.

•**Capítulo 13: Extraer información de sitios web:** Muestra cómo escribir programas que puedan descargar automáticamente páginas web y examinarlas en busca de información. Esto se denomina *web scraping*.

•**Capítulo 14: Hojas de cálculo de Excel:** Aborda la manipulación de hojas de cálculo de Excel con programación para no tener que leerlas, lo cual es útil cuando el número de documentos que hay que analizar se eleva a cientos o miles.

•**Capítulo 15: Hojas de cálculo de Google:** Expone cómo leer y actualizar hojas de cálculo de Google, una popular aplicación basada en web, usando Python.

•**Capítulo 16: Bases de datos SQLite:** Explica cómo utilizar bases

de datos relacionales con SQLite, la pequeña pero potente base de datos de código abierto que viene con Python.

•**Capítulo 17: Documentos de Word y PDF:** Muestra cómo leer documentos de Word y PDF con programación.

•**Capítulo 18: Archivos CSV, JSON y XML:** Sigue explicando cómo usar programación para manipular documentos, en este caso comentando los formatos de serialización de datos CSV, JSON y XML.

•**Capítulo 19: Manejar el tiempo, programar tareas y lanzar programas:** Explica cómo manejan las horas y las fechas los programas de Python y cómo programar el ordenador para que realice tareas a determinadas horas. También muestra cómo nuestros programas de Python pueden lanzar programas que no sean de Python.

•**Capítulo 20: Enviar correos electrónicos, textos y notificaciones push:** Explica cómo escribir programas que nos envíen notificaciones por correo electrónico o al móvil y cómo enviar estos mensajes a otros.

•**Capítulo 21: Hacer gráficos y manipular imágenes:** Explica cómo usar la programación para manipular imágenes, como archivos JPEG o PNG, y trabajar con la librería de creación de gráficos Matplotlib.

•**Capítulo 22: Reconocer texto en imágenes:** Aborda la extracción de texto en imágenes y documentos escaneados para poder procesarlo después con el paquete PyTesseract.

•**Capítulo 23: Controlar el ratón y el teclado:** Explica cómo emplear programación para controlar el ratón y el teclado para automatizar clics y pulsaciones de teclas.

•**Capítulo 24: Motores de texto a voz y reconocimiento de voz:** Expone cómo usar paquetes informáticos avanzados para no solo generar audio a partir de texto, sino también para convertir audio en texto.

Es posible descargar el código fuente y otros recursos para los ejemplos de este libro en la web de Anaya Multimedia (<http://www.anayamultimedia.es>). Para acceder, debemos dirigirnos al botón Selecciona Complemento de la ficha del libro. Puedes utilizar también la web original en <https://nostarch.com/automate-boring-stuff-python-3rd-edition>. Para ver varios de los programas del libro en acción, visita <https://autbor.com/3>.

Descargar e instalar Python

Se puede descargar Python para Windows, macOS y Ubuntu Linux de manera gratuita en <https://python.org/downloads>.

La página de descarga detecta el sistema operativo en uso y recomienda el paquete de descarga adecuado. Hay instaladores no oficiales de Python para los sistemas operativos móviles iOS y Android, pero quedan fuera del ámbito de este libro. Windows, macOS y Linux también cuentan con sus propias opciones de instalación. Descarga el instalador para tu sistema operativo y ejecuta el programa para instalar el software intérprete de Python.

Una nueva versión de Python o del sistema operativo puede cambiar los pasos necesarios para la instalación. Si tienes problemas, consulta <https://autbor.com/install/> para instrucciones actualizadas.

Descarga e instalación de Mu

Aunque el intérprete de Python es el software que ejecuta los programas de Python, el editor Mu es donde introducimos los programas de una manera muy similar a como introducimos texto en un procesador. Puedes descargar Mu en <https://codewith.mu>.

En Windows y macOS, descarga el instalador para tu sistema operativo y ejecútalo haciendo doble clic en el archivo de instalación. Si estás usando macOS, al ejecutar el instalador se abrirá una ventana en la que debes arrastrar el icono de Mu a la carpeta Aplicaciones para continuar con la instalación. Si usas Ubuntu, tendrás que instalar Mu como paquete de Python. En ese caso, haz clic en el botón Instrucciones de la sección

correspondiente de la página de descargas.

Iniciar Mu

Una vez instalado Mu, puedes iniciarlo:

- En Windows, haz clic en el icono Inicio de la esquina inferior izquierda de la pantalla, busca Mu Editor y selecciónalo.
- En macOS, abre la ventana de Finder, haz clic en Aplicaciones y luego en mu-editor. También puedes ejecutar el editor desde Spotlight.
- En Ubuntu, selecciona Aplicaciones > Accesorios > Terminal y escribe **python3 -m mu**.

La primera vez que ejecutes Mu, se abrirá una ventana Seleccionar modo con opciones para Adafruit CircuitPython, BBC micro:bit, Pygame Zero, Python 3 y varios más. Selecciona Python 3. Siempre puedes cambiar el modo después haciendo clic en el botón Modo, en la parte superior de la ventana del editor.

Iniciar IDLE

Este libro utiliza Mu como editor e intérprete de comandos interactivo. No obstante, es posible usar otros editores para escribir código Python. El entorno de desarrollo interactivo IDLE se instala con Python y puede servirte de segundo editor si, por alguna razón, no te funciona Mu. Vamos a iniciar IDLE (suponiendo que tienes instalada la versión Python 3.13):

- En Windows, haz clic en el icono Inicio de la esquina inferior izquierda de la pantalla o escribe **IDLE** en el cuadro de búsqueda y selecciona IDLE (Python 3.13 64-bit).
- En macOS, abre Finder, haz clic en Aplicaciones, luego en Python 3.13 y, por último, en el icono IDLE. También se puede iniciar IDLE desde Spotlight.
- En Ubuntu, selecciona Aplicaciones > Accesorios > Terminal y escribe **idle**. (También puedes hacer clic en el botón Mostrar

aplicaciones de la barra de tareas de Ubuntu y, a continuación, hacer clic en IDLE).

El intérprete de comandos interactivo

Cuando se ejecuta Mu, la ventana que se abre es la ventana del editor de archivos. Se puede abrir el intérprete de comandos interactivo haciendo clic en el botón REPL. Un intérprete de comandos es un programa que permite dar instrucciones al ordenador, como el Terminal o el Símbolo del sistema en macOS y Windows, respectivamente. El software intérprete de Python ejecutará inmediatamente cualquier instrucción que introduzcamos en el intérprete de comandos interactivo de Python.

En Mu, el intérprete de comandos interactivo es un panel en la mitad inferior de la pantalla con algo parecido a este texto:

```
Jupyter QtConsole
```

```
Python 3
```

```
Type 'copyright', 'credits' or 'license' for more information
```

```
IPython -- An enhanced Interactive Python. Type '?' for help.
```

```
In [1]:
```

Si ejecutas IDLE, el intérprete de comandos interactivo es la primera ventana que aparece. Debería estar en blanco, salvo por un texto parecido a este:

```
> > >
```

In [1]: y > > > son *prompts*. Los ejemplos de este libro utilizan el *prompt* > > > para representar el intérprete de comandos interactivo, ya que es más habitual.

Si ejecutas Python desde el Terminal o el Símbolo del sistema, también usarán el *prompt* > > >. In [1]: es una invención de Jupyter Notebook, otro editor de Python popular.

Para ver un ejemplo, escribe lo siguiente en el intérprete de

comandos interactivo, junto al *prompt*:

```
> > > print('Hello, world!')
```

Después de escribir la línea, pulsa **Intro**. El intérprete de comandos interactivo debería mostrar esta respuesta:

```
> > > print('Hello, world!')
```

Hello, world!

¡Acabas de dar una instrucción al ordenador y ha hecho lo que le has mandado!

Cómo buscar ayuda

Los programadores suelen aprender buscando en Internet respuestas a sus preguntas. Es una forma de aprender bastante diferente a lo que está acostumbrada la mayoría, que es con un profesor que enseña y responde preguntas en persona. Lo bueno de usar Internet como aula es que hay muchas comunidades que pueden ayudarte a resolver problemas. De hecho, es probable que tus preguntas ya hayan sido respondidas y esas respuestas estén por ahí esperando a que las encuentres. Si te sale un mensaje de error o tienes problemas para que tu código funcione, no serás la primera persona con ese problema y encontrar una solución es más fácil de lo que piensas.

Por ejemplo, vamos a provocar un error a propósito: escribe '42' + 3 en el intérprete de comandos interactivo. Ahora mismo no te hace falta saber qué significa esa instrucción, pero el resultado debería ser algo así:

```
> > > '42' + 3
```

1 Traceback (most recent call last):

File "<pyshell#0>", line 1, in <module>

```
'42' + 3
```

2 TypeError: can only concatenate str (not "int") to str

> > >

El mensaje de error 2 aparece porque Python no entendía la instrucción. La parte de Traceback 1 del mensaje de error muestra la instrucción específica y el número de línea con el que Python ha tenido problemas. Si no estás seguro de qué hacer con un mensaje de error concreto, búscalo por Internet. Escribe “TypeError: can only concatenate str (not “int”) to str” (incluyendo las comillas) en tu buscador preferido y encontrarás un montón de enlaces explicando qué significa ese error y qué lo causa, como muestra la figura I.2.

Con frecuencia, verás que alguien más ha tenido la misma duda y que otras personas ya la han resuelto. Nadie lo sabe todo sobre la programación, así que parte del trabajo diario de cualquier desarrollador consiste en buscar respuestas a preguntas técnicas.

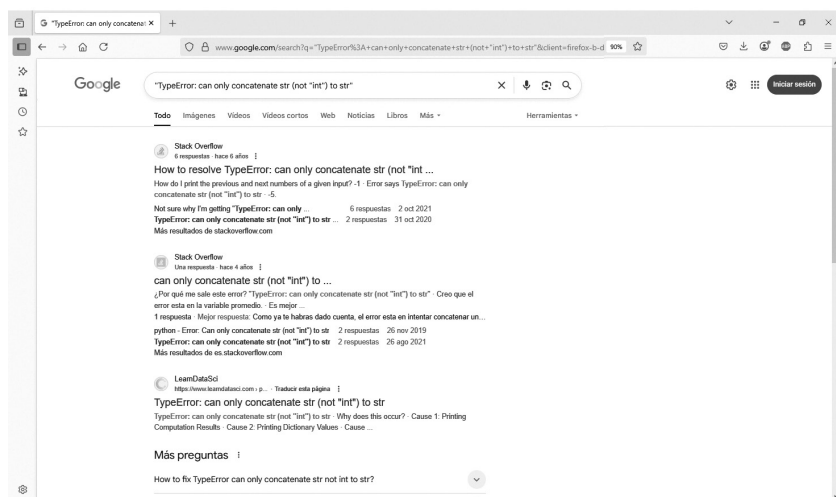


Figura I.2. Los resultados de Google para un mensaje de error pueden ser muy útiles.

Hacer buenas preguntas sobre la programación

Si no encuentras la respuesta buscando en Internet, prueba a preguntar a otros en un foro web como Stack Overflow (<https://stackoverflow.com>) o el subreddit “learn programming”, en <https://>

reddit.com/r/learnprogramming. Pero ten en cuenta que hay formas inteligentes de hacer preguntas de programación que ayudarán a que otros te ayuden. Para empezar, asegúrate de leer las preguntas frecuentes de estos sitios web para ver la manera adecuada de preguntar.

Recuerda lo siguiente a la hora de hacer preguntas de programación:

- Explica lo que estás intentando hacer, no solo lo que has hecho. Esto permite a quien te ayude saber si vas bien o mal.
 - Especifica el punto en el que se ha producido el error. ¿Ha sido al principio del programa o nada más hacer una acción determinada?
 - Copia y pega el mensaje de error entero y tu código en <https://pastebin.com> o <https://gist.github.com>. Estos sitios web hacen que sea más fácil compartir grandes cantidades de código en línea, sin perder nada del formato del texto. Luego puedes poner la URL del código publicado en tu mensaje de correo o foro. Por ejemplo, aquí están algunos de los fragmentos de código que he colgado yo: <https://pastebin.com/2k3LqDsd> y <https://gist.github.com/asweigart/6912168>.
 - Explica lo que has intentado hacer para resolver el problema. Esto dice a los demás que ya has intentado algo para resolver tú el problema.
 - Indica la versión de Python que estás utilizando, y también tu sistema operativo y la versión que ejecutas.
 - Si el error apareció al hacer un cambio en el código, explica exactamente qué cambiaste.
 - Di si puedes reproducir el error cada vez que ejecutas el programa o si se produce solo al realizar ciertas acciones. En este último caso, explica también de qué acciones se trata.
- Sigue siempre las normas de etiqueta en línea. Por ejemplo, no escribas todo en mayúsculas ni hagas demandas poco razonables a la gente que intenta ayudarte.

Encontrarás más información sobre cómo pedir ayuda para programar en el blog <https://autbor.com/help>. Me encanta ayudar a la gente a descubrir Python y escribo tutoriales de programación en mi blog: <https://inventwithpython.com/blog>. También puedes contactar conmigo en al@inventwithpython.com, aunque es posible que obtengas una respuesta más rápida si planteas tus preguntas en <https://reddit.com/r/inventwithpython>.

Novedades en la tercera edición

Esta edición totalmente revisada y actualizada incluye 16 proyectos de programación nuevos para que practiques las habilidades que aprendas. Encontrarás una introducción completa a SQLite y las bases de datos relacionales con el módulo sqlite3 de Python. Explorarás cómo compilar *scripts* de Python como programas ejecutables en Windows, macOS y Linux y cómo crear programas de línea de comandos y ejecutarlos. También aprenderás a realizar operaciones en PDF con PyPDF y PdfMiner, a usar Playwright y Selenium para controlar navegadores web, a hacer que tus programas hablen con librerías de texto a voz, a usar la librería Whisper de OpenAI para crear transcripciones de texto de archivos de sonido y vídeo, a extraer texto de imágenes con PyTesseract, a crear gráficos con matplotlib y a hacer otras muchas cosas.

Resumen

Para la mayoría de la gente, su ordenador es solo un aparato en vez de una herramienta. Pero, aprendiendo a programar, podrás acceder a una de las herramientas más potentes del mundo moderno y te divertirás mucho por el camino. La programación no es neurocirugía, no pasa nada si los aficionados experimentan y cometen errores.

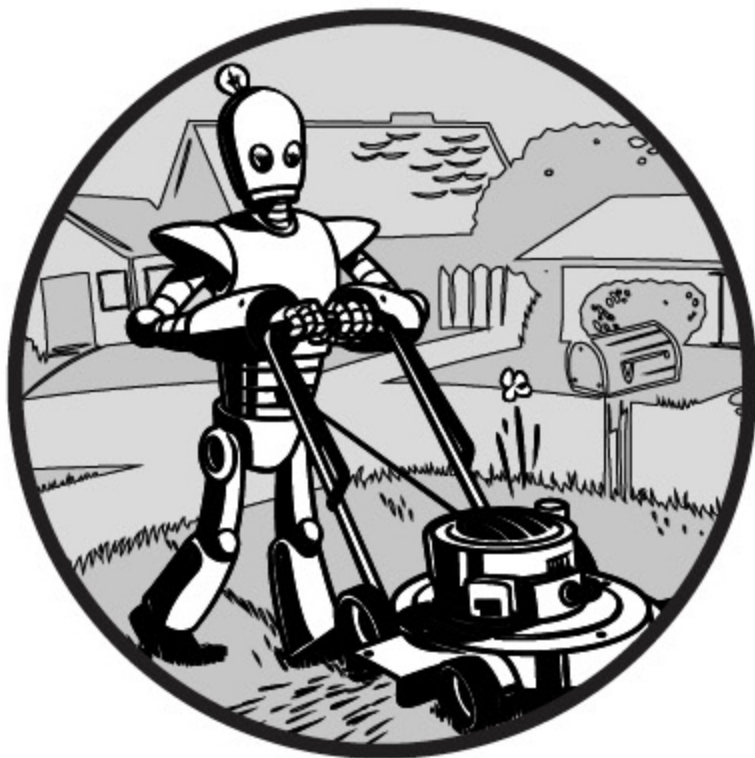
Este libro asume que no tienes ni idea de programación y te enseñará un poco, pero puede que tengas preguntas que van más allá del ámbito de esta obra. Recuerda que hacer buenas preguntas y saber encontrar respuestas serán herramientas muy valiosas en tu viaje por la programación.

¡Vamos allá!

Parte I

Programación básica

Python básico



El lenguaje de programación Python tiene una amplia variedad de construcciones sintácticas, funciones de librería estándar y características de entorno de desarrollo interactivo. Por suerte, puedes ignorar la mayoría de ellas; solo necesitas aprender lo justo para escribir unos programillas útiles.

Sin embargo, para conseguirlo, tendrás que dominar algunos conceptos de programación. Como aprendiz de brujo, tal vez estos conceptos te parezcan tediosos, pero, con un poco de práctica, te permitirán dominar tu ordenador como una varita mágica para hacer cosas increíbles.

Este capítulo contiene unos cuantos ejemplos que te animarán a escribir código en el intérprete de comandos interactivo, también llamado REPL (*read-evaluate-print-loop*, bucle leer-evaluar-imprimir), que permite ejecutar instrucciones de Python de una en una y ver al instante los resultados. Usar el intérprete de comandos interactivo es genial para aprender lo que hacen las instrucciones básicas Python, así que ve probando mientras sigues el libro. Recordarás mejor todo que si te limitas a leer.

Introducir expresiones en el intérprete de comandos interactivo

Puedes ejecutar el intérprete de comandos interactivo iniciando el editor Mu. La introducción de este libro ofrece instrucciones para descargarlo e instalarlo. En Windows, abre el menú Inicio, busca Mu y abre la aplicación. En macOS, abre la carpeta Aplicaciones y haz doble clic en Mu. Haz clic en el botón Nuevo y guarda un archivo en blanco como `blank.py`. Cuando ejecutes este archivo en blanco haciendo clic en el botón Ejecutar o pulsando **F5**, se abrirá el intérprete de comandos interactivo como nuevo panel en la parte inferior de la ventana del editor Mu. Deberías ver un *prompt* `> > >` en el intérprete de comandos interactivo.

También puedes ejecutar el intérprete de comandos interactivo desde el Terminal de línea de comandos (en macOS y Linux) o en Terminal Windows (en Windows, donde también puedes usar el antiguo Símbolo del sistema). Una vez abiertas estas ventanas de línea de comandos, escribe **python** (en Windows) o **python3** (en macOS y Linux). Verás el mismo *prompt* `> > >` para el intérprete de comandos interactivo. Si quieres ejecutar un programa, ejecuta `python` o `python3` seguido del nombre del archivo `.py` del programa, por ejemplo, `python blank.py`. Asegúrate de no ejecutar `python` en el Terminal de macOS porque hacerlo iniciará la versión antigua e incompatible Python 2.7 en determinadas versiones de macOS. Puedes incluso ver un mensaje de advertencia desaconsejando el uso de Python 2.7. En ese caso, sal del intérprete de comandos interactivo 2.7 y ejecuta en su lugar `python3`.

Escribe `2 + 2` en el *prompt* para que Python haga una sencilla suma. La ventana de Mu debería mostrar algo así:

```
> > > 2 + 2
```

```
4
```

```
> > >
```

En Python, $2 + 2$ es una expresión, que es el tipo de instrucción de programación más básico del lenguaje. Las expresiones constan de valores (como 2) y operadores (como +), y siempre pueden evaluarse (es decir, reducirse) a un único valor. Esto significa que podemos usar expresiones en cualquier parte de un código en Python donde también podríamos usar un valor.

En el ejemplo anterior, $2 + 2$ se reduce a un solo valor, 4. Un valor sin operadores también se considera una expresión, aunque se evalúa solo a sí mismo, como aquí:

```
> > > 2
```

```
2
```

El editor Mu tiene un botón REPL que muestra un intérprete de comandos interactivo con un *prompt* `In [1]:`. El popular editor Jupyter Notebook utiliza este tipo de intérprete interactivo, que se puede usar igual que el intérprete de comandos interactivo normal de Python con el *prompt* `> > >`. Los REPL no son exclusivos de Python; muchos lenguajes de programación ofrecen también este tipo de bucle para poder experimentar con el código.

¡Los errores son buenos!

Lo mejor de los ordenadores es que hacen exactamente lo que les decimos que hagan. Eso es también lo peor de los ordenadores. No pueden usar el sentido común para suponer lo que pretendemos hacer. Los programas fallarán si contienen código que el ordenador no entiende, lo que hará que Python muestre un mensaje de error. Los mensajes de error no estropean el ordenador, así que no tengas miedo de cometer errores. Un fallo solo significa que el programa deja de ejecutarse de manera inesperada.

Acostúmbrate a ver mensajes de error, porque te van a salir constantemente, aunque tengas décadas de experiencia

programando. Los mensajes de error son a menudo demasiado vagos y no es fácil que los principiantes los entiendan de primeras. Si quieres saber más sobre un error, puedes buscar el texto exacto del mensaje en Internet para más información. Ten en cuenta que, si estás usando el editor Mu, el método abreviado para copiar al portapapeles el texto destacado en el panel del intérprete de comandos interactivo es **Control-Mayús-C**, mientras que el método para copiar texto en el panel principal del editor es el típico **Control-C**. También puedes echar un vistazo a los recursos del libro para ver una lista de mensajes de error de Python habituales y su significado.

Programar implica algunas operaciones matemáticas con las que tal vez no estés familiarizado:

- La potenciación (o elevación a la potencia de) consiste en multiplicar un número por sí mismo repetidamente, igual que la multiplicación consiste en sumar un número a sí mismo repetidamente. Por ejemplo, dos a la potencia de cuatro (o dos a la cuarta potencia), escrito 2^4 o `2 ** 4`, es el número dos multiplicado por sí mismo cuatro veces: $2^4 = 2 \times 2 \times 2 \times 2 = 16$.
- La aritmética modular es similar al resto de la división. Por ejemplo, $14 \% 4$ se evalúa como 2 porque 14 dividido entre 4 es 3 con un resto de 2. Aunque el operador módulo de Python es `%`, la aritmética modular no tiene nada que ver con los porcentajes.
- La división de enteros sería igual que la división normal, salvo porque el resultado se redondea a la baja. Por ejemplo, $25 / 8$ es 3.125, pero en la división de enteros es 3, y $29 // 10$ es 2 en vez de 2.9.

Podemos usar otros muchos operadores en expresiones de Python. Por ejemplo, la tabla 1.1 recoge todos los operadores matemáticos de Python.

Tabla 1.1. Operadores matemáticos.

Se copió...

Potenciación
Módulo/resto

~~División~~ de enteros

~~División~~

~~Multiplicación~~

~~Respa~~

~~Suma~~

El orden de las operaciones (también llamado precedencia) de los operadores matemáticos de Python es similar al de las matemáticas. El operador `**` se evalúa primero; después, los operadores `*`, `/`, `//` y `%`, de izquierda a derecha; y los operadores `+` y `-` se evalúan los últimos (también de izquierda a derecha). Podemos usar paréntesis para anular la precedencia habitual si es preciso. En Python, los espacios entre los operadores y los valores no son relevantes, excepto el sangrado al principio de la línea. Pero la convención, o regla no oficial, es dejar un solo espacio entre operadores y valores. Escribe la siguiente expresión en el intérprete de comandos interactivo:

```
> > > 2 + 3 * 6
```

20

```
> > > (2 + 3) * 6
```

30

```
> > > 48565878 * 578453
```

28093077826734

```
> > > 2 * 8
```

256

```
> > > 23 / 7
```

3.2857142857142856

```
> > > 23 / 7
```

3

```
> > > 23 % 7
```

```
2
```

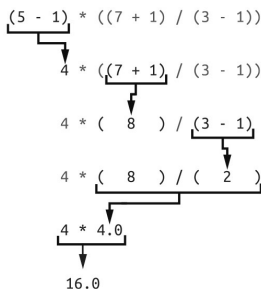
```
> > > 2 + 2
```

```
4
```

```
> > > (5 - 1) * ((7 + 1) / (3 - 1))
```

```
16.0
```

En cada caso, tú, como programador, debes introducir la expresión, pero Python se encarga de evaluarla y seguirá evaluando partes de la expresión hasta convertirla en un solo valor:



Estas reglas para juntar operadores y valores para formar expresiones son parte fundamental de Python como lenguaje de programación, igual que las reglas gramaticales que nos ayudan a comunicarnos. Aquí tenemos un ejemplo:

Esta es una oración gramaticalmente correcta en español.

Esta gramaticalmente es oración no en español correcta una.

La segunda línea es difícil de analizar sintácticamente porque no sigue las reglas del español. Del mismo modo, si escribimos mal una instrucción, Python no la entenderá y mostrará un mensaje de error `SyntaxError`, como aquí:

```
> > > 5 +
```

File "<python-input-0>", line 1

```
5 +
```

```
^
```

SyntaxError: invalid syntax

```
> > > 42 + 5 + * 2
```

File "<python-input-0>", line 1

```
42 + 5 + * 2
```

```
^
```

SyntaxError: invalid syntax

Siempre puedes probar si una instrucción funciona introduciéndola en el intérprete de comandos interactivo. No te preocupes porque no se va a romper el ordenador, lo peor que puede pasar es que Python responda con un mensaje de error. Los desarrolladores de software profesionales reciben mensajes de error cada dos por tres mientras escriben código.

Los tipos de datos enteros, flotantes y cadenas

Recuerda que las expresiones son simplemente valores combinados con operadores y siempre se reducen a un único valor. Un tipo de dato es una categoría para los valores y cada valor pertenece a un tipo de dato exactamente. La tabla 1.2 recoge los tipos de datos más habituales en Python. Los valores -2 y 30, por ejemplo, son valores enteros. El tipo de datos entero (o int) indica valores que son números enteros. Los números con decimales, como 3.14, se denominan flotantes (o float). Observa que, aunque el valor 42 sea un entero, el valor 42.0 sería un flotante. Los programadores utilizan a menudo la palabra número para referirse a enteros y flotantes de manera conjunta, aunque número no es en sí un tipo de datos de Python.

Un detalle sutil sobre Python es que cualquier operación

matemática realizada con un entero y un flotante da como resultado un flotante, no un entero. Aunque $3 + 4$ da como resultado el entero 7, el resultado de la expresión $3 + 4.0$ es el flotante 7.0. Cualquier división entre dos enteros con el operador `/` también produce un resultado flotante. Por ejemplo, $16 / 4$ se evalúa como 4.0, no 4. La mayor parte del tiempo, esta información no es relevante para nuestros programas, pero explica por qué los números pueden tener de repente decimales.

Tabla 1.2. Tipos de datos habituales.

	Ejemplo
Entero (<code>int</code>)	2, 3, 4, 5
Flotante (<code>float</code>)	0.5, 0.0, 0.5, 1.0, 1.25
Cadena (<code>str</code>)	'¡Hola!', '11 gatos', '5'

Los programas de Python también pueden tener valores de texto, llamados cadenas, o `str`. Pon siempre comillas simples (`'`) alrededor de una cadena (como en `'Hola'` o `'¡Adiós, mundo cruel!'`) para que Python sepa dónde termina y acaba la cadena. Existen las cadenas que no tienen caracteres dentro, se llaman cadenas en blanco o vacías y quedan así: `''`. Veremos las cadenas con más detenimiento en el capítulo 8.

Es posible que alguna vez te salga el mensaje de error `SyntaxError: unterminated string literal`, como en este ejemplo:

```
> > > 'Hello, world!
```

```
SyntaxError: unterminated string literal (detected at line 1)
```

Este error significa que es probable que hayas olvidado la comilla simple al final de la cadena.

Concatenación y replicación de cadenas

El significado de un operador puede cambiar en función del tipo de dato de los valores que lo acompañan. Por ejemplo, `+` es el operador de suma cuando va con dos valores enteros o flotantes. En cambio, cuando se usa para combinar dos valores de cadena, une las cadenas como operador de concatenación. Escribe lo siguiente

en el intérprete de comandos interactivo:

```
> > > 'Alice' + 'Bob'
```

```
'AliceBob'
```

La expresión se reduce a un nuevo valor de cadena único que combina el texto de las dos cadenas. Sin embargo, si intentamos usar el operador `+` con una cadena y un entero, Python no sabrá cómo manejar eso y mostrará un mensaje de error:

```
> > > 'Alice' + 42
```

Traceback (most recent call last):

File "<python-input-0>", line 1, in <module>

```
'Alice' + 42
```

TypeError: can only concatenate str (not "int") to str

El mensaje de error `can only concatenate str (not "int") to str` significa que Python pensaba que estábamos intentando concatenar un entero a la cadena `'Alice'`. El código tendrá que convertir explícitamente el entero en una cadena porque Python no puede hacerlo automáticamente. (Enseguida explicaré como convertir un tipo de datos en otro y hablaré de las funciones `str()`, `int()` y `float()`).

El operador `*` multiplica dos valores enteros o flotantes, pero, cuando se usa con un valor de cadena y un valor entero, se convierte en el operador de replicación de cadena. Introduce una cadena multiplicada por un número en el intérprete de comandos interactivos para ver qué pasa:

```
> > > 'Alice' * 5
```

```
'AliceAliceAliceAliceAlice'
```

La expresión se reduce a un único valor de cadena que repite la cadena original un número de veces igual al valor integral. La replicación de cadenas es un truco útil, pero no se utiliza tanto como la concatenación de cadenas.

El operador `*` solo puede emplearse con dos valores numéricos (para la multiplicación) o con un valor de cadena y un entero (para la replicación de cadenas). Si no se hace así, Python mostrará un mensaje de error como este:

```
> > > 'Alice' * 'Bob'
```

Traceback (most recent call last):

File "<python-input-0>", line 1, in <module>

```
'Alice' * 'Bob'
```

TypeError: can't multiply sequence by non-int of type 'str'

```
> > > 'Alice' * 5.0
```

Traceback (most recent call last):

File "<python-input-0>", line 1, in <module>

```
'Alice' * 5.0
```

TypeError: can't multiply sequence by non-int of type 'float'

Es normal que Python no entienda estas expresiones: no se puede multiplicar dos palabras y es difícil replicar una cadena arbitraria un número fraccionario de veces.

Las expresiones, los tipos de datos y los operadores pueden parecerte demasiado abstractos ahora mismo, pero, a medida que domines estos conceptos, podrías crear programas cada vez más sofisticados que hagan cálculos con datos extraídos de hojas de cálculo, sitios web, la salida de otros programas y otras cosas.

Almacenar valores en variables

Una variable es como una caja en la memoria del ordenador donde se puede guardar un único valor. Si queremos usar el resultado de una expresión evaluada más adelante en un programa, podemos guardarla en una variable.

Los valores se almacenan en variables con una declaración de asignación, que consiste en un nombre de variable, un signo de igualdad (denominado operador de asignación) y el valor que hay que almacenar. En la declaración de asignación `spam = 42`, una variable llamada `spam` tendrá almacenado el valor entero 42.

Puedes imaginar una variable como una caja etiquetada en la que se mete un valor, aunque en el capítulo 6 veremos cómo una etiqueta unida al valor podría ser una metáfora mejor. Ambas se representan en la figura 1.1.

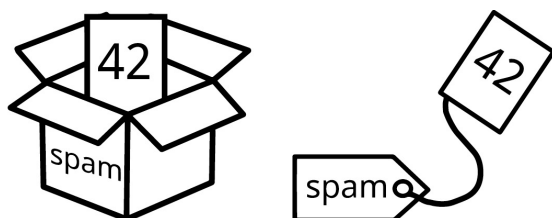


Figura 1.1. El código `spam = 42` es como decir al programa: “la variable `spam` tiene ahora el valor entero 42”.

Por ejemplo, introduce lo siguiente en el intérprete de comandos interactivo:

```
1 >>> spam = 40
```

```
>>> spam
```

```
40
```

```
>>> eggs = 2
```

```
2 >>> spam + eggs
```

```
42
```

```
>>> spam + eggs + spam
```

```
82
```

```
3 >>> spam = spam + 2
```

```
>>> spam
```

42

Una variable se inicializa (o crea) la primera vez que se almacena un valor en ella 1. Después, podemos usarla en expresiones con otras variables y valores 2. Cuando se asigna un nuevo valor a una variable 3, se olvida el anterior, por eso al final del ejemplo la evaluación de spam es 42 en vez de 40. Esto se llama sobrescribir la variable. Introduce el siguiente código en el intérprete de comandos interactivo para intentar sobrescribir una cadena:

```
>>> spam = 'Hello'
```

```
>>> spam
```

'Hello'

```
>>> spam = 'Goodbye'
```

```
>>> spam
```

'Goodbye'

Igual que la caja de la figura 1.1, la variable spam de la figura 1.2 almacena 'Hello' hasta que reemplazamos la cadena por 'Goodbye'.

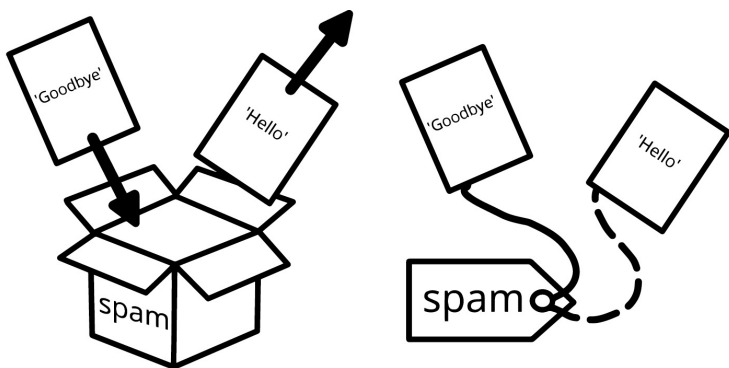


Figura 1.2. Cuando se asigna un valor nuevo a una variable, se olvida el anterior.

Puedes pensar en sobrescribir una variable como reasignar la etiqueta con el nombre a un valor nuevo.

Nombres de variables

Un buen nombre de variable describe los datos que contiene. Imagina que te mudas a una casa nueva y etiquetas todas las cajas de la mudanza como “Cosas”. ¡Luego no encontrarás nada! La mayoría de los ejemplos de este libro (y de la documentación de Python) utilizan nombres de variables genéricos como `spam`, `eggs` y `bacon`, que vienen del *sketch* cómico *Spam* de los Monty Python. Pero, en tus programas, los nombres descriptivos te ayudarán a hacer tu código más legible.

Aunque puedes poner casi cualquier nombre a una variable, Python tiene algunas restricciones de nomenclatura. El nombre de una variable debe cumplir las siguientes cuatro reglas:

- No puede contener espacios.
- Solo puede usar letras, números y guiones bajos (`_`).
- No puede empezar por un número.
- No puede ser una palabra clave de Python, como `if`, `for`, `return` o cualquiera de las palabras clave que aprenderás en este libro.

La tabla 1.3 muestra ejemplos de nombres de variables lícitos.

Tabla 1.3. Nombres de variables válidos y no válidos.

Nombres variables válidos

`current_balance` (no se puede usar guiones altos).

`currentBalance` (no se puede usar espacios).

`4account` (no puede empezar por un número).

`42` (puede empezar por guion bajo, pero no por un número).

`TOTAL_$UM` (los caracteres especiales como `$` no están permitidos).

`hello'` (los caracteres especiales como `'` no están permitidos).

Los nombres de variable distinguen mayúsculas y minúsculas, lo

que significa que spam, SPAM, Spam y sPaM son cuatro variables diferentes. Es convención de Python que las variables empiecen con minúscula: spam en vez de Spam.

Opiniones sobre el estilo del código y PEP 8

En las ediciones anteriores de este libro se utilizaba el estilo *camelCase* en vez de guiones bajos para separar palabras en los nombres de variables; esto es, una variable podía llamarseDeEsteModo en vez de llamarse_de_este_modo. Este segundo estilo se denomina *snake_case* porque los guiones bajos entre las palabras parecen culebrillas (mientras que las mayúsculas del *camelCase* parecen las jorobas del camello). Algunos programadores experimentados podrían señalar que la guía de estilo de código oficial de Python, PEP 8, dice que habría que usar los guiones bajos. Yo prefiero usar sin complejos el *camelCase* y baso mi defensa en la sección “A Foolish Consistency Is the Hobgoblin of Little Minds” de la propia PEP 8:

Es importante mantener la coherencia con la guía de estilo. Pero lo más importante es saber cuándo no ser coherente: a veces, la guía de estilo simplemente no se aplica. En caso de duda, utiliza tu mejor criterio.

A los ordenadores les da igual el estilo que usemos y la PEP 8 no son los diez mandamientos. No importa qué estilo se use, siempre que se haga con consistencia. Para demostrarlo, he reescrito el código de este libro en *snake_case* porque sirve de las dos maneras.

Tu primer programa

Aunque el intérprete de comandos interactivo está bien para ejecutar instrucciones de Python de una en una, para escribir programas enteros necesitaremos el editor de archivos. Este editor es similar a otros editores de texto como Notepad y TextMate, pero cuenta con algunas características específicas para introducir código fuente. Para abrir un archivo nuevo en Mu, haz clic en el botón Nuevo de la parte superior.

La pestaña que se abre debería contener un cursor esperando a que escribas, pero es distinto del intérprete de comandos interactivo, que ejecuta instrucciones de Python en cuanto pulsas **Intro**. El

editor de archivos permite introducir varias instrucciones, guardar el archivo y, después, ejecutar el programa. Así es como se diferencian:

- El intérprete de comandos interactivo siempre será el que presenta el *prompt* `> > >` o `In [1]:`.

- El editor de archivos no tiene los *prompts* `> > >` ni `In [1]:`.

¡Ha llegado el momento de crear tu primer programa! Cuando se abra la ventana del editor de archivos, escribe lo siguiente:

```
# Este programa saluda y pregunta el nombre.
```

```
print("Hello, world!")
```

```
print("What is your name?") # Pregunta el nombre.
```

```
my_name = input('>')
```

```
print("It is good to meet you, ' + my_name)
```

```
print("The length of your name is:")
```

```
print(len(my_name))
```

```
print("What is your age?") # Pregunta la edad.
```

```
my_age = input('>')
```

```
print("You will be ' + str(int(my_age) + 1) + ' in a year.")
```

Una vez introducido el código fuente, guárdalo para no tener que volver a escribirlo cada vez que inicies Mu. Haz clic en Guardar, escribe **hello.py** en el campo Nombre y haz clic en Guardar. Deberías guardar los programas de vez en cuando mientras los escribes. Así, si falla el ordenador o sales por accidente de Mu, no perderás todo el código. Como atajo, puedes pulsar **Control-G** en Windows y Linux o **Comando-S** en macOS para guardar tu archivo.

Una vez guardado el programa, vamos a ejecutarlo. Pulsa la tecla **F5** o haz clic en el botón Ejecutar. Escribe tu nombre cuando te lo

pida el programa. La salida del programa en el intérprete de comandos interactivo debería ser algo parecido a esto:

```
Hello, world!
```

```
What is your name?
```

```
> Al
```

```
It is good to meet you, Al
```

```
The length of your name is:
```

```
2
```

```
What is your age?
```

```
> 4
```

```
You will be 5 in a year.
```

```
> > >
```

Cuando no hay más líneas de código que ejecutar, el programa termina, es decir, deja de ejecutarse. (También se puede decir que se sale del programa). El editor Mu muestra el *prompt* > > > del intérprete de comandos interactivo cuando el programa ha terminado por si queremos añadir más código Python.

Puedes cerrar el editor de archivos haciendo clic en la X de la pestaña del archivo como cuando se cierra una pestaña en un navegador. Para volver a cargar un programa guardado, haz clic en Cargar en el menú. Hazlo ahora y, en la ventana que aparecerá, selecciona hello.py y haz clic en el botón Abrir. El programa hello.py que acabamos de guardar debería abrirse en la ventana del editor de archivos.

Puedes ver la ejecución paso a paso de un programa con la herramienta de visualización Python Tutor, en <http://pythontutor.com>. Haz clic en el botón de avance para moverte por cada paso de la ejecución del programa. Podrás ver cómo cambian los valores de las variables y la salida.

Disección del programa

Con el nuevo programa abierto en el editor de archivos, vamos a echar un vistazo rápido a las instrucciones de Python que utiliza mirando lo que hace cada línea de código.

Comentarios

La siguiente línea es un comentario:

```
# Este programa saluda y pregunta el nombre.
```

Python ignora los comentarios, por lo que podemos usarlos para escribir notas o recordatorios sobre lo que pretende hacer el código. Cualquier texto del resto de la línea que empieza con una almohadilla (#) forma parte del comentario.

A veces, los programadores ponen una almohadilla delante de una línea de código para anularla temporalmente mientras prueban un programa. Esto se llama deshabilitar con comentarios y puede ser útil cuando se intenta averiguar por qué no funciona un código. Luego se puede quitar la almohadilla para devolver la línea al código.

Python también ignora la línea en blanco después del comentario. Podemos añadir tantas líneas en blanco como queramos al programa. Este espacio puede facilitar la lectura del código, igual que pasa con los párrafos de un libro.

La función print()

La función print() muestra en la pantalla el valor de la cadena dentro de sus paréntesis:

```
print("Hello, world!")
```

```
print("What is your name?") # Pregunta el nombre.
```

La línea print("Hello, world!") significa "Imprime el texto de la cadena 'Hello, world!'". Cuando Python la ejecuta, decimos que está llamando a la función print() y el valor de la cadena se pasa a la

función. Un valor que se pasa a una llamada a una función es un argumento. Observa que las comillas no aparecen en la pantalla; simplemente, marcan dónde empieza y acaba la cadena, no forman parte del texto del valor de la cadena.

Nota *También se puede usar esta función para mostrar una línea en blanco en la pantalla, llamando a `print()` y dejando los paréntesis vacíos.*

Cuando escribimos un nombre de función, los paréntesis de apertura y cierre del final lo identifican como el nombre de una función. Por eso, en este libro verás `print()` en vez de `print`. Es una convención no dejar espacios entre el nombre de la función y el paréntesis de apertura, aunque Python no lo exige. El capítulo 3 describe las funciones en mayor detalle.

La función `input()`

La función `input()` espera que el usuario escriba texto con el teclado y pulse **Intro**:

```
my_name = input('>')
```

Esta llamada a función se evalúa en una cadena idéntica al texto del usuario y el resto del código asigna la variable `my_name` a este valor de cadena. La cadena `'>'` pasada a la función hace que aparezca el *prompt* `>`, que sirve al usuario como indicador de que se espera que escriba algo. El programa no tiene por qué pasar una cadena a la función `input()`; si la llamamos, esperará el texto del usuario sin mostrar *prompt* alguno.

Los prompts `>` y `>>>`

Se puede pasar cualquier cadena a `input()` para cambiar el *prompt* que aparece al ejecutar el programa. La llamada a `input('>')` pone un paréntesis angular en la pantalla a modo de *prompt*. Esto es diferente del *prompt* `>>>` que aparece en el intérprete de comandos interactivo de Python. En este libro, `>>>` indica una respuesta al intérprete de comandos interactivo de Python y `>` una respuesta a un programa que ejecuta la llamada `input('>')`. Mi elección de `'>'` fue arbitraria, se puede usar cualquier *prompt* o

ninguno.

Puedes pensar en la llamada a la función `input()` como una expresión que se evalúa como cualquier cadena que escriba el usuario. Si el usuario escribe 'Al', la declaración de asignación sería `my_name = 'Al'`.

Si al llamar a `input()` aparece un error, como `NameError: name 'Al' is not defined`, el problema es que se está ejecutando el código con Python 2 en vez de Python 3.

El mensaje de saludo

La llamada a `print()` contiene la expresión 'It is good to meet you, ' + `my_name` entre paréntesis:

```
print('It is good to meet you, ' + my_name)
```

Recuerda que las expresiones siempre pueden evaluarse como un único valor. Si 'Al' es el valor almacenado en `my_name`, esta expresión se evalúa a 'It is good to meet you, Al'. Este único valor de cadena se pasa a `print()`, que lo imprime en la pantalla.

La función `len()`

Podemos pasar a la función `len()` un valor de cadena (o una variable que contiene una cadena), y la función se evalúa como el valor entero del número de caracteres de esa cadena:

```
print('The length of your name is:')
```

```
print(len(my_name))
```

Escribe lo siguiente en el intérprete de comandos interactivo para probar esto:

```
> > > len('hello')
```

5

```
> > > len('My very energetic monster just scarfed nachos.')
```

```
> > > len("")
```

```
0
```

Igual que en esos ejemplos, `len(my_name)` se evalúa como un entero. Decimos que la llamada a la función `len()` devuelve este valor entero y ese valor es el valor de retorno de la llamada a la función. Después se pasa a `print()` para que lo muestre en la pantalla. La función `print()` permite pasarle valores enteros o de cadena, pero fíjate en el error que aparece al escribir lo siguiente en el intérprete de comandos interactivo:

```
> > > print('I am ' + 29 + ' years old.')
```

Traceback (most recent call last):

File "<python-input-0>", line 1, in <module>

```
print('I am ' + 29 + ' years old.')
```

TypeError: can only concatenate str (not "int") to str

La función `print()` no está produciendo el error, sino que lo hace la expresión que hemos intentado pasar a `print()`. Obtendremos el mismo mensaje de error si escribimos la expresión directamente en el intérprete de comandos interactivo:

```
> > > 'I am ' + 29 + ' years old.'
```

Traceback (most recent call last):

File "<python-input-0>", line 1, in <module>

```
'I am ' + 29 + ' years old.'
```

TypeError: can only concatenate str (not "int") to str

Python da error porque el operador `+` se puede usar solo para sumar dos números o concatenar dos cadenas. No se puede sumar un entero a una cadena porque eso no está permitido en Python.

Podemos resolverlo utilizando una versión de cadena del entero, como veremos en la siguiente sección.

Las funciones `str()`, `int()` y `float()`

Si queremos concatenar un entero, como 29, con una cadena para pasarlos a `print()`, necesitamos obtener el valor '29', que es la forma en cadena de 29. Se puede pasar un valor entero a la función `str()` para que devuelva una versión en cadena del entero, así:

```
> > > str(29)
```

```
'29'
```

```
> > > print('I am ' + str(29) + ' years old.')
```

```
I am 29 years old.
```

Dado que `str(29)` se evalúa como '29', la expresión `'I am ' + str(29) + ' years old.'` da como resultado `'I am ' + '29' + ' years old.'`, lo que a su vez produce `'I am 29 years old.'`. Este es el valor de cadena que se pasa a la función `print()`.

Las funciones `str()`, `int()` y `float()` evaluarán el valor que se les pase como cadena, entero y flotante, respectivamente. Prueba a convertir algunos valores en el intérprete de comandos interactivo con estas funciones y observa qué sucede:

```
> > > str(0)
```

```
'0'
```

```
> > > str(-3.14)
```

```
'-3.14'
```

```
> > > int('42')
```

```
42
```

```
> > > int('-99')
```

-99

```
> > > int(1.25)
```

1

```
> > > int(1.99)
```

1

```
> > > float('3.14')
```

3.14

```
> > > float(10)
```

10.0

Los ejemplos anteriores llaman a las funciones `str()`, `int()` y `float()` y les pasan valores de los otros tipos de datos para obtener una cadena, un entero o un flotante a partir de ellos.

La función `str()` es útil cuando tenemos un flotante o un entero que queremos concatenar con una cadena. La función `int()` también puede ayudarnos si tenemos un número como cadena que tenemos que usar para algún cálculo. Por ejemplo, la función `input()` siempre devuelve una cadena, incluso aunque el usuario escriba un número. Escribe `spam = input()` en el intérprete de comandos interactivo y después escribe 101 cuando espere tu texto:

```
> > > spam = input()
```

101

```
> > > spam
```

'101'

El valor guardado en `spam` no es el entero 101, sino la cadena '101'. Si queremos hacer cuentas con el valor de `spam`, tendremos que usar la función `int()` para obtener ese valor en su forma entera y guardar este como el nuevo valor de la variable. Si `spam` es la

cadena '101', la expresión `int(spam)` se evaluará como el valor entero 101 y la declaración de asignación `spam = int(spam)` equivaldrá a `spam = 101`:

```
> > > spam = int(spam)
```

```
> > > spam
```

```
101
```

Ahora deberíamos poder tratar la variable `spam` como un entero en vez de como una cadena:

```
> > > spam * 10 / 5
```

```
202.0
```

Ten en cuenta que, si pasamos a `int()` un valor que no pueda evaluarse como entero, Python mostrará un mensaje de error:

```
> > > int('99.99')
```

Traceback (most recent call last):

File "<python-input-0>", line 1, in <module>

```
int('99.99')
```

ValueError: invalid literal for int() with base 10: '99.99'

```
> > > int('twelve')
```

Traceback (most recent call last):

File "<python-input-0>", line 1, in <module>

```
int('twelve')
```

ValueError: invalid literal for int() with base 10: 'twelve'

La función `int()` también es útil para redondear hacia abajo un valor flotante:


```
> > > int(7.7)
```

7

```
> > > int(7.7) + 1
```

8

Hemos usado las funciones `int()` y `str()` en las últimas tres líneas del programa para obtener un valor del tipo de datos adecuado para el código:

```
print('What is your age?') # Pregunta la edad.
```

```
my_age = input('>')
```

```
print('You will be ' + str(int(my_age) + 1) + ' in a year.')
```

La variable `my_age` contiene el valor devuelto por `input()`. Dado que la función `input()` siempre devuelve una cadena (aunque el usuario introduzca un número), podemos usar el código `int(my_age)` para devolver un valor entero de la cadena de `my_age`. Luego se suma 1 a este valor entero en la expresión `int(my_age) + 1`.

Equivalencia entre texto y número

Aunque el valor de cadena de un número se considera un valor completamente diferente de la versión entera o flotante, un entero puede ser igual a un flotante:

```
> > > 42 = '42'
```

False

```
> > > 42 = 42.0
```

True

```
> > > 42.0 = 0042.000
```

True

Python hace esta distinción porque las cadenas son texto, mientras

que los enteros y los flotantes son números.

El resultado de esta suma se pasa a la función `str()`: `str(int(my_age) + 1)`. El valor de cadena devuelto se concatena con las cadenas 'You will be ' e ' in a year.' para evaluarse como un valor de cadena grande. Esta gran cadena se pasa finalmente a `print()` para mostrarlo en la pantalla.

Supongamos que el usuario introduce la cadena '4' para `my_age`. Los pasos de evaluación serían algo así:



```
print('You will be ' + str(int(myAge) + 1) + ' in a year.')
print('You will be ' + str(int( '4' ) + 1) + ' in a year.')
print('You will be ' + str(    4 + 1    ) + ' in a year.')
print('You will be ' + str(        5        ) + ' in a year.')
print('You will be ' +          '5'          + ' in a year.')
print('You will be 5'                    + ' in a year.')
print('You will be 5 in a year.')
```

La cadena '4' se convierte en un entero para poder sumarle 1. El resultado es 5. La función `str()` convierte el resultado de nuevo en una cadena para poder concatenarla con la segunda cadena, 'in a year.', y crear el mensaje final.

La función `type()`

Enteros, flotantes y cadenas no son los únicos tipos de datos en Python. A medida que sigas aprendiendo sobre la programación, es posible que te encuentres con valores de otro tipo. Siempre puedes pasarlos a la función `type()` para determinar qué son. Por ejemplo, escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > type(42)
```

```
<class 'int'>
```

```
> > > type(42.0)
```

```
<class 'float'>
```

```
> > > type('forty two')
```

```
<class 'str'>
```

```
> > > name = 'Zophie'
```

```
> > > type(name) # La variable name tiene un valor de tipo  
cadena.
```

```
<class 'str'>
```

```
> > > type(len(name)) # La función len() devuelve valores  
enteros.
```

```
<class 'int'>
```

No solo podemos pasar cualquier valor a `type()`, sino que (como con cualquier llamada a una función) también podemos pasarle cualquier variable o expresión para determinar el tipo de dato como el que se evalúa. La función `type()` en sí devuelve valores, pero los paréntesis angulares significa que no son código Python válido sintácticamente, no se puede ejecutar un código como `spam = <class 'str'>`.

Las funciones `round()` y `abs()`

Vamos a descubrir otras dos funciones de Python que, al igual que `len()`, toman un argumento y devuelven un valor. La función `round()` acepta un valor flotante y devuelve el entero más próximo. Escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > round(3.14)
```

```
3
```

```
> > > round(7.7)
```

```
8
```

```
> > > round(-2.2)
```

```
-2
```

La función `round()` también acepta un segundo argumento opcional especificando cuántos decimales debería redondear. Escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > round(3.14, 1)
```

```
3.1
```

```
> > > round(7.7777, 3)
```

```
7.778
```

El comportamiento para redondear números medios es un poco raro. La llamada `round(3.5)` redondea hacia arriba, a 4, mientras que `round(2.5)` redondea hacia abajo, a 2. Para los números intermedios que terminan en .5, el número se redondea al entero par más cercano. Esto se denomina redondeo a la par.

La función `abs()` devuelve el valor absoluto del argumento numérico. En matemáticas, esto se define como la distancia desde 0, pero para mí es más fácil pensar en ello como la forma positiva del número. Escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > abs(25)
```

```
25
```

```
> > > abs(-25)
```

```
25
```

```
> > > abs(-3.14)
```

```
3.14
```

```
> > > abs(0)
```

```
0
```

Python incluye varias funciones diferentes sobre las que hablaremos en este libro. Este apartado muestra cómo experimentar con ellas en el intérprete de comandos interactivo para ver cómo se comportan con distintas entradas. Es una técnica habitual para practicar el nuevo código que aprendas.

Cómo guardan los ordenadores datos con números binarios

Ya es suficiente código Python por ahora. En este punto, puede que estés pensando que la programación parece casi magia. ¿Cómo sabe el ordenador transformar $2 + 2$ en 4? La respuesta es demasiado complicada para este libro, pero puedo explicar parte de lo que pasa entre bambalinas comentando qué tienen que ver los números binarios (números que solo contienen los dígitos 1 y 0) con la informática.

El hackeo en las películas suele implicar secuencias de unos y ceros que fluyen por la pantalla. Esto parece misterioso e impresionante, pero ¿qué significan realmente esos unos y ceros? La respuesta es que el binario es el sistema numérico más sencillo y se puede implementar con componentes baratos para el hardware informático. El binario, también llamado sistema numérico de base 2, puede representar todos los números que nuestro sistema numérico decimal de base 10, con el que estamos más familiarizados, puede representar. El sistema decimal tiene diez dígitos, del 0 al 9. La tabla 1.4 muestra los primeros 27 números enteros en decimal y en binario.

Tabla 1.4. Números equivalentes en decimal y binario.

Binario

00010
10001
20100
30001
40010
50101
60000
70000
80000

Imagina estos sistemas numéricos como un cuentakilómetros mecánico, como en la figura 1.3. Cuando llegas al último dígito, cada uno se reinicia en 0 y se incrementa el valor del siguiente dígito. En decimal, el último dígito es 9 y, en binario, 1. Por eso el número decimal después de 9 es 10 y el número decimal después de 999 es 1000. Del mismo modo, el número binario después de 1 es 10 y el número binario después de 111 es 1000. Sin embargo, 10 en binario no representa la misma cantidad que el diez decimal, sino que representa dos. Y 1000 en binario no es mil, sino ocho. Puedes ver un cuentakilómetros interactivo binario y decimal en <https://inventwithpython.com/odometer>

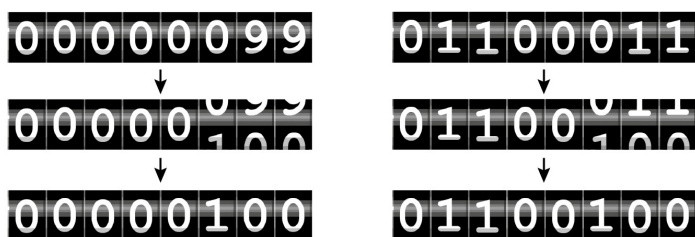


Figura 1.3. Un cuentakilómetros mecánico en decimal (izquierda) y binario (derecha).

Representar números binarios con hardware informático es más fácil que representar números decimales porque solo hay dos estados que representar. Por ejemplo, los discos de Blu-ray y DVD tienen superficies lisas y hendiduras grabadas en su superficie que reflejan o no el láser del reproductor de discos, respectivamente. Los circuitos pueden tener corriente eléctrica circulando por ellos o no. Todos estos estándares de hardware tienen formas de representar dos estados diferentes. Por otro lado, sería caro crear componentes electrónicos de alta calidad que fueran lo suficientemente sensibles como para detectar la diferencia entre 10 niveles de tensión diferentes con una precisión fiable. Es más económico utilizar componentes sencillos, y dos estados binarios son lo más sencillo que se puede conseguir.

Estos dígitos binarios se denominan bits para abreviar. Un solo bit puede representar dos números y 8 bits (o 1 byte) pueden representar 28 o 256 números entre 0 y 255 en decimal, o entre 0 y 11111111 en binario. Esto es similar a cómo un único dígito

decimal puede representar 10 números (de 0 a 9), mientras que un número digital con ocho dígitos puede representar 108 o cien millones de números (de 0 a 99.999.999). Los archivos del ordenador se miden en función de cuántos bytes ocupan:

- Un kilobyte (KB) son 210 o 1.024 bytes.
- Un megabyte (MB) son 220 o 1.048.576 bytes (o 1.024 KB).
- Un gigabyte (GB) son 230 o 1.073.741.824 bytes (o 1.024 MB).
- Un terabyte (TB) son 240 o 1.099.511.627.776 bytes (o 1.024 GB).

El texto de *Romeo y Julieta*, de Shakespeare, supone aproximadamente 135 KB. Una foto de alta resolución tiene entre 2 MB y 5 MB. Una película puede estar entre 1 GB y 50 GB, dependiendo de la duración y la calidad de la imagen. Sin embargo, los fabricantes de discos duros y memorias externas mienten descaradamente sobre lo que significan estos términos. Por ejemplo, al llamar un TB a 1.000.000.000.000 bytes en vez de 1.099.511.627.776 bytes, pueden promocionar un disco duro de 9,09 TB como 10 TB.

Los unos y ceros del sistema binario no solo pueden representar un entero, sino también cualquier forma de datos. En lugar de 0 a 255, un byte puede representar los números entre -128 y 127 con un sistema llamado complementos a dos. Los números flotantes fraccionarios pueden representarse en binario con un sistema denominado IEEE-754.

Se puede guardar texto en los ordenadores como números binarios asignando a cada letra, signo de puntuación o símbolo un número único. Un sistema para representar texto como números recibe el nombre de codificación. La codificación más popular para texto es UTF-8. En UTF-8, una A mayúscula se representa con el número decimal 65 (o 01000001 como número binario de 8 bits), un signo de cierre de interrogación (?) se representa con el número 63, y el carácter numérico 7 se representa con el número 55. La cadena 'Hello' se almacena como los números 72, 101, 108, 108 y 111. Al guardarse en un ordenador, "Hello" aparece como una secuencia de bits: 0100100001100101011011000110110001101111.

¡Vaya! ¡Como las de las películas de *hackers*!

Los ingenieros necesitan inventar una manera de codificar cada forma de datos como números. Las fotos y las imágenes se pueden descomponer en una cuadrícula bidimensional de cuadros de colores llamados píxeles. Cada píxel puede usar tres bytes para representar cuánto rojo, verde y azul contiene. En el capítulo 21 hablaremos con más detenimiento sobre los datos de imagen, pero, por ver un pequeño ejemplo, los números 255, 0 y 255 podrían representar un píxel con la cantidad máxima de rojo y azul, pero cero verde, lo que dará como resultado un píxel morado.

El sonido se compone de ondas de aire comprimido que llegan a nuestros oídos y que el cerebro interpreta como sensaciones acústicas. Podemos representar gráficamente la intensidad y la frecuencia de estas ondas en el tiempo. Los números de ese gráfico pueden entonces convertirse a números binarios y almacenarse en un ordenador, que después controlará los altavoces para reproducir el sonido. Esto es una simplificación de cómo funciona el sonido en el ordenador, pero explica cómo los números puede representar la *Sinfonía n.º 5* de Beethoven.

Los datos de varias imágenes se combinan con datos de audio para guardar vídeos. Todas las formas de información pueden codificarse en números binarios. Naturalmente, se puede profundizar mucho en esto, pero básicamente así es cómo la amplia variedad de datos en la era de la información se representa con unos y ceros.

Resumen

Podemos calcular expresiones con una calculadora o introducir concatenaciones de cadenas con un procesador de texto. También podemos replicar cadenas fácilmente copiando y pegando texto. Pero las expresiones y los valores que las componen (operadores, variables y funciones) son los bloques de construcción básicos para hacer programas. Una vez que sepas manejar estos elementos, podrás mandar a Python que opere con grandes cantidades de datos por ti.

Te resultará útil recordar los distintos tipos de operadores (+, -, *,

/, //, % y ** para operaciones matemáticas y + y * para operaciones con cadenas) y los tres tipos de datos (enteros, flotantes y cadenas) presentados en este capítulo.

También hemos visto unas pocas funciones. Las funciones `print()` e `input()` manejan salida (a la pantalla) y la entrada (desde el teclado) de texto simple. La función `len()` toma una cadena y la evalúa como un entero del número de caracteres de la cadena. Las funciones `str()`, `int()` y `float()` se evaluarán como formas de cadenas, enteros o flotantes del valor que se les pase. La función `round()` devuelve el entero redondeado y `abs()` el valor absoluto de los argumentos.

En el próximo capítulo veremos cómo decir a Python que tome decisiones inteligentes sobre qué código ejecutar, qué código saltarse y qué código repetir basándose en los valores que tenga. Esto se conoce como control de flujo y permite escribir programas que tomen decisiones inteligentes.

Preguntas prácticas

1. ¿Cuáles de los siguientes son operadores y cuáles valores?

*

'hello'

-88.8

-

/

+

5

2. De la siguiente expresión, ¿qué es una variable y qué es una cadena?

spam

'spam'

3. Nombra tres tipos de datos.

4. ¿De qué se compone una expresión? ¿Qué hacen todas las expresiones?

5. Este capítulo ha presentado las declaraciones de asignación, como `spam = 10`. ¿Cuál es la diferencia entre una expresión y una declaración?

6. ¿Qué contiene la variable `bacon` después de ejecutar el código?

```
bacon = 20
```

```
bacon + 1
```

7. ¿Cómo deberían evaluarse estas dos expresiones?

```
'spam' + 'spamspam'
```

```
'spam' * 3
```

8. ¿Por qué `eggs` es un nombre de variable válido y `100` no lo es?

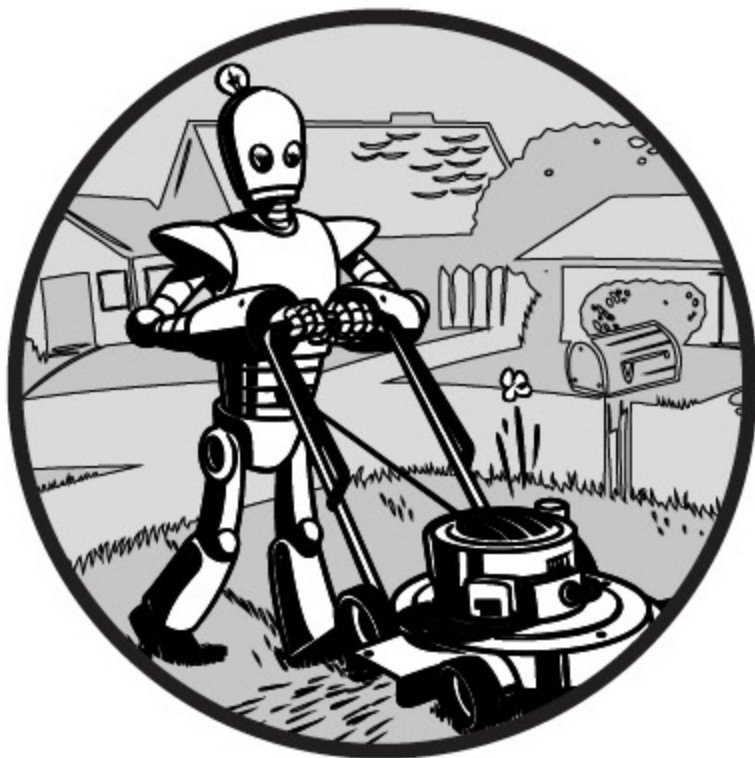
9. ¿Qué tres funciones pueden utilizarse para obtener la versión entera, flotante o de cadena de un valor?

10. ¿Por qué esta expresión provoca un error? ¿Cómo podemos arreglarlo?

```
'I eat ' + 99 + ' burritos.'
```

Para subir nota: Busca en Internet la documentación de Python para la función `len()`. Estará en una página web titulada “Funciones integradas”. Lee por encima la lista de funciones que tiene Python, busca lo que hacen `bin()` y `hex()` y experimenta con ellas en el intérprete de comandos interactivo.

if-else y el control de flujo



Bueno, ya conoces lo básico de las instrucciones de un programa y sabes que un programa es solo una serie de instrucciones. Pero el verdadero poder de la programación no es simplemente ejecutar una instrucción tras otra como una lista de tareas para el fin de semana. En función de cómo se evalúen las expresiones, un programa puede decidir saltarse instrucciones, repetirlas o elegir una entre varias para ejecutar. De hecho, casi nunca queremos que los programas empiecen por la primera línea de código y vayan ejecutando cada línea hasta el final. Las declaraciones de control de flujo pueden decidir qué instrucciones ejecuta Python en qué condiciones.

Estas declaraciones de control de flujo se corresponden directamente con los símbolos de un diagrama de flujo, así que veremos versiones en diagrama de flujo de los programas de los que hablaremos en este capítulo. La figura 2.1 muestra un diagrama de flujo para qué hacer si está lloviendo. Sigue el camino trazado por las flechas de principio a fin.

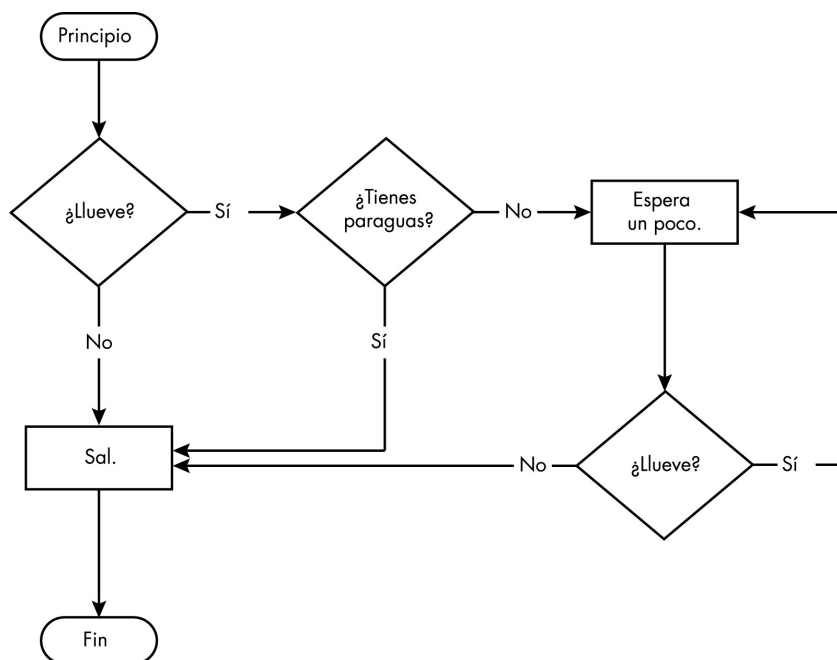


Figura 2.1. Un diagrama de flujo que nos dice qué hacer si está lloviendo.

En un diagrama de flujo, suele haber más de una manera de llegar del principio al final. Lo mismo sucede con las líneas de código de un programa informático. Los diagramas de flujo representan estos puntos de bifurcación con diamantes, los otros pasos con rectángulos y el principio y el final con rectángulos de esquinas redondeadas.

Antes de aprender sobre las declaraciones de control de flujo, hay que saber cómo representar esas opciones de sí o no y entender cómo escribir esos puntos de bifurcación en código Python. Para conseguirlo, vamos a explorar los valores booleanos, los

comparadores de comparación y los operadores booleanos.

Valores booleanos

Mientras que los tipos de datos entero, flotante y cadena tienen un número ilimitado de valores posibles, el tipo de datos booleano solo tiene dos valores: True y False (verdadero y falso). Cuando se introducen como código Python, los valores booleanos True y False van sin las comillas propias de las cadenas y siempre empiezan por mayúscula, con el resto de la palabra en minúsculas. Observa que estos valores no llevan comillas porque son distintos de los valores de cadena 'True' y 'False'. Escribe lo siguiente en el intérprete de comandos interactivo:

```
1 >>> spam = True
```

```
>>> spam
```

```
True
```

```
2 >>> true
```

Traceback (most recent call last):

File "<python-input-0>", line 1, in <module>

NameError: name 'true' is not defined

```
3 >>> False = 2 + 2
```

File "<python-input-0>", line 1, in <module>

SyntaxError: can't assign to False

Algunas de estas instrucciones son incorrectas deliberadamente y harán que aparezcan mensajes de error. Como ocurre con cualquier otro valor, podemos usar valores booleanos en expresiones y almacenarlos en variables 1. Si no utilizas la mayúscula y minúscula bien 2 o si tratas de utilizar True y False como nombres de variables 3, Python dará error.

Los operadores de comparación, también llamados operadores relacionales, comparan dos valores y se reducen a un único valor booleano. La tabla 2.1 recoge los operadores de comparación.

Tabla 2.1. Operadores de comparación.

Significado

Igual que se evalúa como True.

$4 == 2 + 2$ se evalúa como True.

No igual que se evalúa como True.

$'Hello' != 'Hello'$ se evalúa como False.

Menor que se evalúa como False.

$1.999 < 5$ se evalúa como True.

Mayor que se evalúa como False.

$99 > 4 + 8$ se evalúa como True.

Menor o igual que se evalúa como True.

$5 <= 5$ se evalúa como True.

Mayor o igual que se evalúa como True.

$5 >= 5$ se evalúa como True.

Estos operadores se evalúan como True o False dependiendo de los valores que se les den. Vamos a probar algunos operadores, empezando con $==$ y $!=$:

```
>>> 42 == 42
```

True

```
>>> 42 == 99
```

False

```
>>> 2 == 3
```

True

```
>>> 2 == 2
```

False

Como cabría esperar, $==$ (igual que) se evalúa como True cuando los valores a ambos lados son iguales y $!=$ (no igual que) se evalúa

como True cuando los dos valores son diferentes. Los operadores `==` y `!=` pueden funcionar con valores de cualquier tipo de datos:

```
> > > 'hello' == 'hello'
```

True

```
> > > 'hello' == 'Hello'
```

False

```
> > > 'dog' == 'cat'
```

True

```
> > > True == True
```

True

```
> > > True == False
```

True

```
> > > 42 == 42.0
```

True

```
1 > > > 42 == '42'
```

False

Observa que un valor entero o flotante nunca será igual que un valor de cadena. La expresión `42 == '42'` se evalúa como False porque Python considera que el entero 42 es distinto de la cadena '42'. Sin embargo, Python considera que el entero 42 es lo mismo que el flotante 42.0.

Los operadores `<`, `>`, `<=` y `>=`, por otro lado, funcionan correctamente solo con valores enteros y valores flotantes:

```
> > > 42 < 100
```

True

> > > 42 > 100

False

> > > 42 < 42

False

1 > > > eggs = 42

> > > eggs <= 42

True

> > > my_age = 29

2 > > > my_age >= 10

True

Con frecuencia, utilizarás operadores de comparación para comparar valores de variables con otros valores, como en los ejemplos `eggs <= 42` 1 y `my_age >= 10` 2, o para comparar entre sí los valores de dos variables. (De todos modos, comparar dos valores literales como `'dog' != 'cat'` siempre tiene el mismo resultado). Veremos más ejemplos de esto al hablar sobre las declaraciones de control de flujo.

La diferencia entre los operadores `=` y `=`

Es posible que hayas observado que el operador `==` (igual que) tiene dos signos de igualdad, mientras que el operador `=` (asignación) tiene solo uno. Es fácil confundir estos operadores, así que recuerda esto:

- El operador `==` pregunta si dos valores son iguales entre sí.
- El operador `=` pone el valor de la derecha en la variable de la izquierda.

Para recordar cuál es cuál, fíjate en que el operador `==` (igual que) consta de dos caracteres, igual que el operador `!=` (no igual

que).

Operadores booleanos

Los tres operadores booleanos (and, or y not) sirven para comparar valores booleanos. Al igual que los operadores de comparación, reducen las expresiones a un valor booleano. Vamos a ver estos operadores en detalle, empezando por and.

El operador and siempre toma dos valores (o expresiones) booleanos, así que se considera un operador booleano binario. El operador and evalúa una expresión como True si ambos valores son True; si no, se evalúa como False. Introduce algunas expresiones con and en el intérprete de comandos interactivo para verlo en acción:

```
> > > True and True
```

```
True
```

```
> > > True and False
```

```
False
```

Una tabla de verdad muestra todos los resultados posibles de un operador booleano. La tabla 2.2 es la tabla de verdad para el operador and.

Tabla 2.2. La tabla de verdad del operador and.

Expresión	Resultado
-----------	-----------

True and True	True
---------------	------

True and False	False
----------------	-------

False and True	False
----------------	-------

False and False	False
-----------------	-------

Al igual que el operador and, el operador or también toma dos valores (o expresiones booleanas) y, por tanto, también se considera un operador binario. Sin embargo, or evalúa una expresión como True si cualquiera de los dos valores booleanos es True. Si ambos son False, evalúa como False:

> > > **False or True**

True

> > > **False or False**

False

La tabla 2.3 es la tabla de verdad de or y muestra todos los resultados posibles del operador.

Tabla 2.3. La tabla de verdad del operador or.

Expresión como...

True or True

True or False

~~False~~ or True

False or False

A diferencia de and y or, not opera solo con un valor (o expresión) booleano, lo que lo convierte en un operador unitario. El operador not se evalúa simplemente como el valor booleano opuesto:

> > > **not True**

False

1 > > > **not not not not True**

True

Como al hablar y escribir, se pueden usar muchos operadores not 1, pero nunca hay motivo para hacer eso en un programa real. La tabla 2.4 es la tabla de verdad de not.

Tabla 2.4. La tabla de verdad del operador not.

Expresión como...

~~False~~ True

~~True~~ False

Combinar operadores booleanos y de comparación

Dado que los operadores de comparación se evalúan como valores booleanos, podemos utilizarlos en expresiones con los operadores booleanos.

Recuerda que los operadores and, or y not se denominan operadores booleanos porque siempre operan con los valores booleanos True y False. Aunque expresiones como $4 < 5$ no son valores booleanos, son expresiones que se reducen a valores booleanos. Prueba a introducir algunas expresiones booleanas con operadores de comparación en el intérprete de comandos interactivo:

```
> > > (4 < 5) and (5 < 6)
```

True

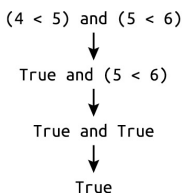
```
> > > (4 < 5) and (9 < 6)
```

False

```
> > > (1 = 2) or (2 = 2)
```

True

El ordenador evaluará primero la expresión de la izquierda y, a continuación, la de la derecha. Cuando conozca el valor booleano para cada una, reducirá la expresión completa a un valor booleano. Puedes pensar en el proceso de evaluación del ordenador para $(4 < 5)$ y $(5 < 6)$ como lo siguiente:



También podemos usar varios operadores booleanos en una expresión junto con los operadores de comparación:

```
> > > spam = 4
```

```
> > > 2 + 2 = spam and not 2 + 2 = (spam + 1) and 2 * 2 =  
2 + 2
```

True

Los operadores booleanos siguen un orden de operaciones, igual que los matemáticos. Después de la evaluación de cualquier operador matemático o de comparación, Python evalúa primero los operadores not, después and y, por último, or.

Componentes del control de flujo

Las declaraciones de control de flujo empiezan a menudo con una parte llamada condición y siempre van seguidas de un bloque de código denominado proposición. Antes de hablar de las declaraciones de control de flujo específicas de Python, vamos a ver en qué consisten las condiciones y los bloques.

Condiciones

Las expresiones booleanas que hemos visto hasta ahora podrían considerarse todas condiciones, que es lo mismo que expresiones; “condición” es solo un nombre más específico en el contexto de las declaraciones de control de flujo. Una condición siempre se evalúa como un valor booleano, True o False. Una declaración de control de flujo decide qué hacer basándose en si su condición es True o False, y casi todas las declaraciones de control de flujo utilizan una condición. Con frecuencia, escribirás código que podría describirse en lenguaje normal como: “si esta condición es verdadera, haz esto; si no, haz esto otro”. Otro código que escribirás equivale a decir: “sigue repitiendo estas instrucciones siempre y cuando esta condición sea verdadera”.

Bloques de código

Las líneas de código de Python se pueden agrupar en bloques y podemos ver dónde empieza y acaba un bloque por el sangrado de las líneas. Hay cuatro reglas para los bloques:

- Un bloque nuevo empieza cuando aumenta el sangrado.
- Un bloque puede contener otros bloques.

- Un bloque acaba cuando disminuye el sangrado a cero o al sangrado del bloque que lo contiene.
- Python espera un bloque nuevo inmediatamente después de cualquier declaración que termine con dos puntos.

Los bloques se entienden mejor fijándose en el sangrado del código, así que vamos a buscar los bloques en parte del pequeño programa que vemos aquí:

```
username = 'Mary'

password = 'swordfish'

if username == 'Mary':

    1 print('Hello, Mary')

    if password == 'swordfish':

        2 print('Access granted.')

    else:

        3 print('Wrong password.')
```

El primer bloque de código 1 empieza en la línea `print('Hello, Mary')` y contiene todas las líneas de después. Dentro de este bloque, hay otro 2, que solo tiene una línea en `print('Access granted.')`. El tercer bloque 3 también ocupa solo una línea, `print('Wrong password.')`.

Ejecución de programas

En el programa `hello.py` del capítulo anterior, Python empezaba a ejecutar instrucciones desde el principio del programa e iba bajando de una en una. La ejecución del programa (o ejecución, sin más) es un término para la instrucción que se está ejecutando. Si pones el dedo en cada línea de la pantalla cuando se ejecuta, puedes ver tu dedo como la ejecución del programa.

Sin embargo, no todos los programas se ejecutan simplemente

bajando. Si usas el dedo para seguir un programa con declaraciones de control de flujo, es probable que acabes moviéndolo por distintos sitios del código fuente en función de las condiciones.

Las declaraciones de control de flujo

Vamos a explorar la parte más importante del control de flujo: las propias declaraciones. Las declaraciones representan los diamantes que veíamos en el diagrama de flujo de la figura 2.1 y son las decisiones reales que tomarán los programas.

if

El tipo más común de declaración de control de flujo es la declaración if. La proposición de una declaración if (es decir, el bloque que la sigue) se ejecutará si la condición de la declaración es True. La proposición se omite si la condición es False.

En lenguaje natural, una declaración if se leería así: “Si esta condición es verdadera, ejecuta el código de la proposición”. En Python, una declaración if consta de lo siguiente:

- La palabra clave if.
- Una condición (es decir, una expresión que se evalúa como True o False).
- Dos puntos.
- A partir de la siguiente línea, un bloque de código sangrado (llamado proposición o bloque if).

Por ejemplo, supongamos que tenemos un código que comprueba si alguien se llama Alice:

```
name = 'Alice'  
  
if name == 'Alice':  
  
    print('Hi, Alice.')
```

Todas las declaraciones de control de flujo terminan con dos puntos y van seguidas de un nuevo bloque de código (la proposición). La proposición de esta declaración `if` es el bloque `print('Hi, Alice.')`. La figura 2.2 muestra cómo sería el diagrama de flujo de este código.

Prueba a cambiar el nombre de la variable por otra cadena distinta de 'Alice' y vuelve a ejecutar el programa. Observa que no aparece "Hi, Alice" porque ese código se ha omitido.

`else`

Una declaración `if` puede ir seguida de una declaración `else`, que se ejecuta solo cuando la condición de la declaración `if` es `False`. En lenguaje natural, una declaración `else` se leería así: "Si esta condición es verdadera, ejecuta este código. De otro modo, ejecuta ese código".

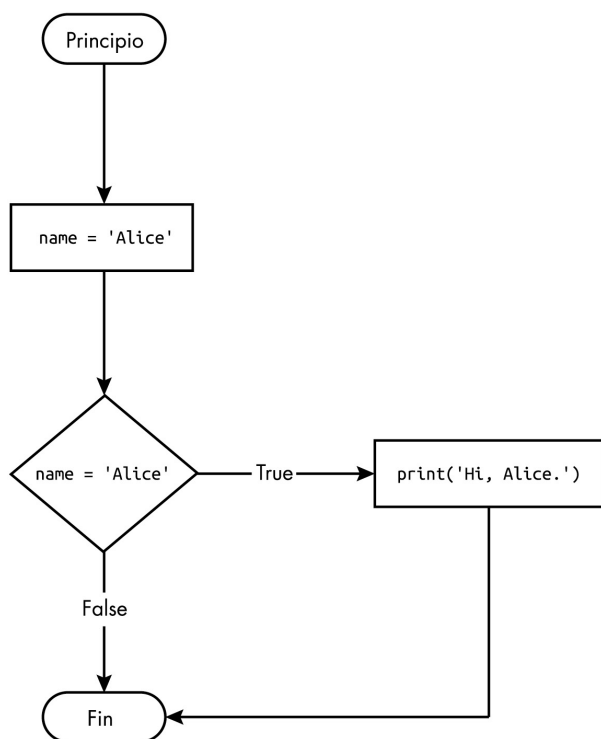


Figura 2.2. Diagrama de flujo para una declaración `if`.

La declaración `else` no tiene condición y su código consta de lo siguiente:

- La palabra clave `else`.
- Dos puntos.
- A partir de la siguiente línea, un bloque de código sangrado (llamado proposición o bloque `else`).

Volviendo al ejemplo de Alice, vamos a echar un vistazo a un código que utiliza una declaración `else` para ofrecer un saludo diferente si la persona no se llama Alice:

```
name = 'Alice'

if name == 'Alice':

    print('Hi, Alice.')

else:

    print('Hello, stranger.')
```

La figura 2.3 muestra cómo sería el diagrama de flujo de este código.

Cambia el nombre de la variable a otra cadena que no sea `'Alice'` y ejecuta de nuevo el programa. En vez de `'Hi, Alice.'`, verás en la pantalla `'Hello, stranger.'`.

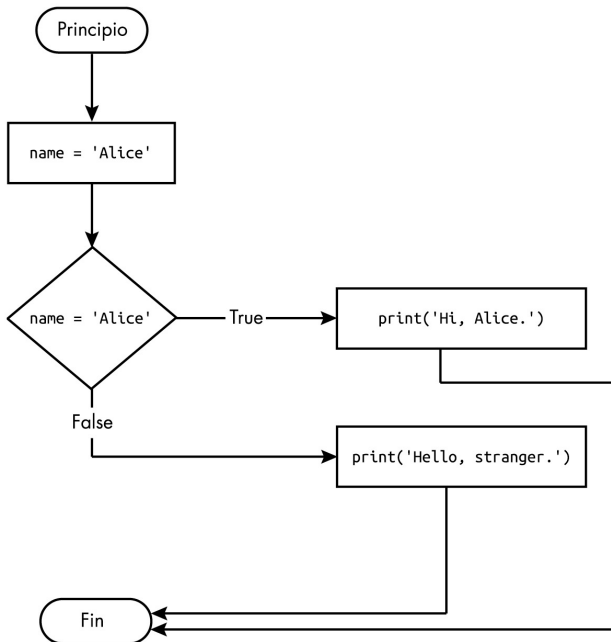


Figura 2.3. Diagrama de flujo para una declaración else.

elif

Utilizamos if o else cuando queremos que se ejecute solo una de dos proposiciones, pero puede haber casos en los que necesitemos que se ejecute una entre varias proposiciones posibles. La declaración elif es una declaración “else if” que siempre va detrás de una declaración if u otra elif. Da otra condición que se comprueba solo si todas las anteriores son False. En código, una declaración elif consta de:

- La palabra clave elif.
- Una condición (es decir, una expresión que se evalúa como True o False).
- Dos puntos.
- A partir de la siguiente línea, un bloque de código sangrado (llamado proposición o bloque elif).

Vamos a añadir una elif al controlador de nombres para ver esta declaración en acción:

```
name = 'Alice'
```

```
age = 33
```

```
if name == 'Alice':
```

```
    print('Hi, Alice.')
```

```
elif age < 12:
```

```
    print('You are not Alice, kiddo.')
```

Esta vez, el programa comprueba la edad de la persona y le dice algo diferente si tiene menos de 12 años. La figura 2.4 muestra el diagrama de flujo correspondiente.

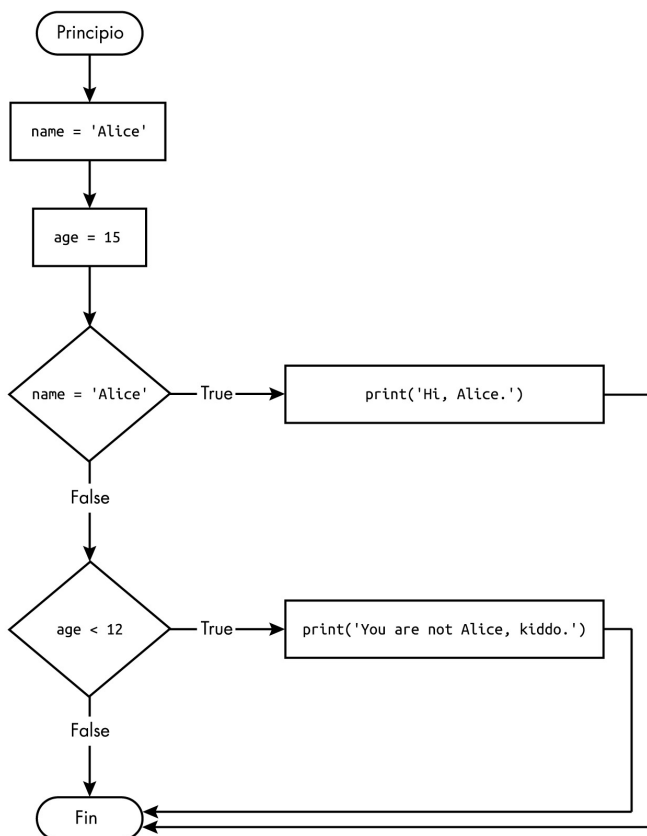


Figura 2.4. Diagrama de flujo para una declaración elif.

La proposición elif se ejecuta si `age < 12` es `True` y `name == 'Alice'` es `False`. Sin embargo, si ambas condiciones son `False`, Python omite las dos proposiciones. No hay garantía de que se ejecute al menos una de las proposiciones; en una cadena de declaraciones elif, solo se ejecutará una o ninguna. Cuando una de las condiciones de la declaración resulta ser `True`, el resto de las proposiciones elif se omiten automáticamente. Para ver un ejemplo, abre una nueva ventana del editor de archivos e introduce el siguiente código, guardándolo como `vampire.py`:

```
name = 'Carol'
```

```
age = 3000
```

```
if name == 'Alice':  
    print('Hi, Alice.')  
  
    elif age < 12:  
        print('You are not Alice, kiddo.')  
  
    elif age > 2000:  
        print('Unlike you, Alice is not an undead, immortal vampire.')  
  
    elif age > 100:  
        print('You are not Alice, grannie.')
```

Aquí, hemos añadido dos declaraciones elif más para hacer que el comprobador de nombres salude a una persona con distintas respuestas basándose en su edad. La figura 2.5 muestra el diagrama de flujo para esto.

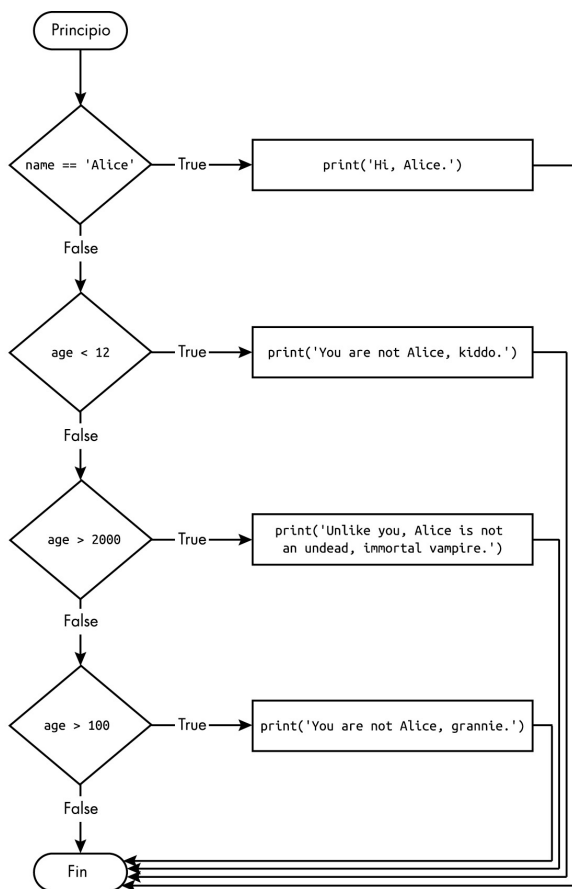


Figura 2.5. Diagrama de flujo para varias declaraciones elif en el programa vampire.py.

El orden de las declaraciones elif es importante. Vamos a reorganizarlas para introducir un error. Recuerda que el resto de las proposiciones elif se omiten automáticamente en cuanto se encuentra una condición True, así que, si cambiamos alguna de las proposiciones en vampire.py, vamos a tener un problema. Cambia el código de la siguiente manera y guárdalo como vampire2.py:

```
name = 'Carol'
```

```
age = 3000
```

```
if name == 'Alice':
```

```
print('Hi, Alice.')

elif age < 12:

print('You are not Alice, kiddo.')

1 elif age > 100:

print('You are not Alice, grannie.')

elif age > 2000:

print('Unlike you, Alice is not an undead, immortal vampire.')
```

Supongamos que la variable `age` contiene el valor 3000 antes de que se ejecute este código. Cabría esperar que el código mostrase la cadena 'Unlike you, Alice is not an undead, immortal vampire.'. Sin embargo, como la condición `age > 100` es `True` (después de todo, 3.000 es mayor que 100) 1, se imprime la cadena 'You are not Alice, grannie.' y el resto de las declaraciones `elif` se omiten automáticamente. Recuerda que solo se ejecutará una de las proposiciones como mucho, así que el orden es importante para las declaraciones `elif`.

La figura 2.6 muestra el diagrama de flujo para el código anterior. Observa cómo los diamantes para `age > 100` y `age > 2000` están cambiados.

Otra opción es colocar una declaración `else` detrás de la última `elif`. De ese modo, nos aseguramos de que se ejecute al menos una (y solo una) de las proposiciones. Si las condiciones de todas las declaraciones `if` y `elif` son `False`, se ejecuta la proposición `else`. A modo de ejemplo, vamos a recrear el programa de Alice para utilizar proposiciones `if`, `elif` y `else`:

```
name = 'Carol'

age = 3000

if name == 'Alice':

print('Hi, Alice.')
```

elif age < 12:

print('You are not Alice, kiddo.')

else:

print('You are neither Alice nor a little kid.')

La figura 2.7 muestra el diagrama de flujo para este nuevo código, que guardaremos como littleKid.py.

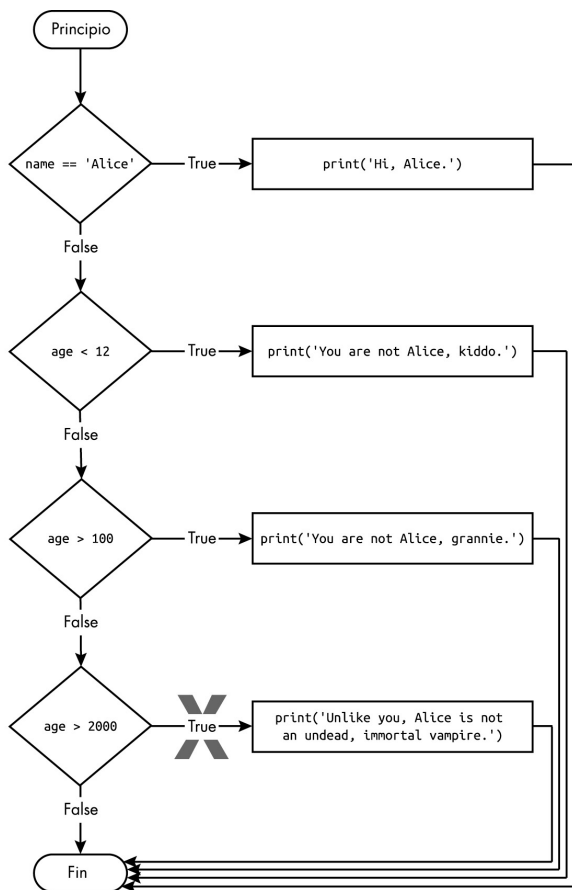


Figura 2.6. Diagrama de flujo del programa vampire2.py. El camino tachado con la X nunca se ejecutará lógicamente porque si la edad fuese mayor que 2.000 ya se habría determinado como mayor que 100.

En lenguaje natural, este tipo de estructura de control de flujo sería así: “Si la primera condición es verdadera, haz esto. Si no, si la segunda condición es verdadera, haz aquello. Si no, haz otra cosa”. Cuando utilices declaraciones `if`, `elif` y `else` juntas, recuerda estas reglas sobre cómo ordenarlas para evitar errores como el de la figura 2.6. En primer lugar siempre hay una declaración `if`; cualquier declaración `elif` que necesitemos debería ir detrás de la `if`. Segundo, si quieres asegurarte de que se ejecute al menos una proposición, cierra la estructura con una declaración `else`.

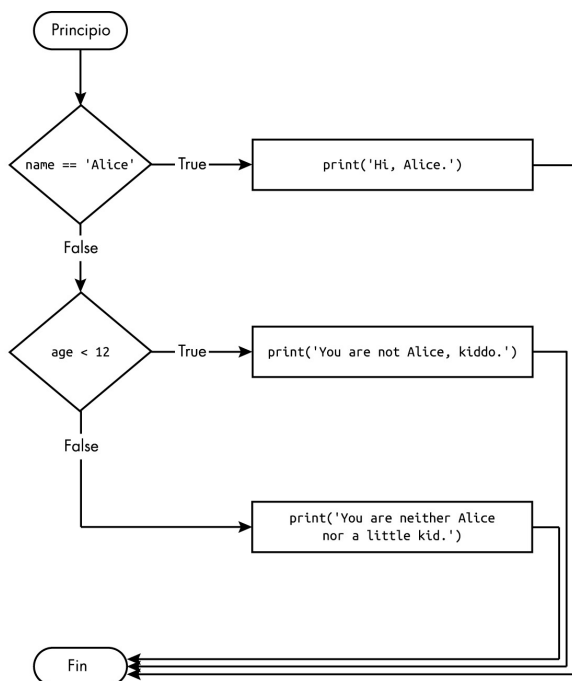


Figura 2.7. Diagrama de flujo para el programa `littleKid.py`.

Como ves, las declaraciones de control de flujo pueden hacer los programas más sofisticados, pero también más complicados. No te desesperes, te irás encontrando cada vez más cómodo con esta complejidad a medida que practiques escribiendo código. Todos los programadores han perdido en algún momento una hora averiguando por qué su programa no funciona por cosas como haber puesto por accidente `<` en vez de `<=`. Todos cometemos errores así.

Un programa corto: el Día de todo al revés

Con tus conocimientos de los valores booleanos y las declaraciones if-else, escribe el siguiente código en un archivo nuevo y guárdalo como `oppositeday.py`:

```
today_is_opposite_day = True

# Establece say_it_is_opposite_day basándose en
today_is_opposite_day:

1 if today_is_opposite_day == True:

say_it_is_opposite_day = True

else:

say_it_is_opposite_day = False

# Si es el Día de todo al revés, cambia a say_it_is_opposite_day:

if today_is_opposite_day == True:

2 say_it_is_opposite_day = not say_it_is_opposite_day

# Dice qué día es:

if say_it_is_opposite_day == True:

print("Today is Opposite Day.")

else:

print("Today is not Opposite Day.")
```

Ejecutar este programa produce el resultado “Today is not Opposite Day.”. Hay dos variables en este código. Al principio del programa, la variable `today_is_opposite_day` está establecida en `True`. La siguiente declaración if comprueba si esta variable es `True` (lo es) 1 y establece la variable `say_it_is_opposite_day` en `True`; de lo contrario, la establecería en `False`. La segunda declaración if comprueba si `today_is_opposite_day` está establecida en `True` (sigue

estándolo) y, si es el caso, el código cambia (establece el valor booleano contrario) la variable 2. Por último, la tercera declaración `if` comprueba si `say_it_is_opposite_day` es `True` (no lo es) y muestra ‘Today is Opposite Day.’; de lo contrario, mostraría ‘Today is not Opposite Day.’.

Si cambiamos la primera línea del programa por `today_is_opposite_day = False` y lo volvemos a ejecutar, seguirá mostrando ‘Today is not Opposite Day.’. Si repasamos el programa, podemos suponer que las primeras declaraciones `if-else` establecen `say_it_is_opposite_day` en `False`. La condición de la segunda declaración `if` es `False`, así que omite ese bloque de código. Por último, la condición de la tercera declaración `if` es también `False` y muestra ‘Today is not Opposite Day.’.

Así pues, si hoy no es el Día de todo al revés, el programa indica correctamente ‘Today is not Opposite Day.’. Y, si es el Día al revés, el programa muestra (también correctamente) ‘Today is not Opposite Day.’ como diríamos si fuese el Día de todo al revés. Lógicamente, este programa nunca imprimirá ‘Today is Opposite Day.’ al margen de que la variable de `if` esté establecida en `True` o en `False`. En realidad, podríamos reemplazar este programa entero por solo `print(“Today is not Opposite Day.”)` y sería el mismo programa. Por eso no debería pagarse a los programadores por línea de código escrito.

Un programa corto: una calculadora de capacidad falsa

En el capítulo 1, comentaba que los fabricantes de discos duros y unidades de memoria flash mienten sobre las capacidades anunciadas de sus productos utilizando una definición diferente de TB y GB. Vamos a escribir un programa para calcular en qué medida sus capacidades anunciadas son engañosas. Escribe el siguiente código en un archivo nuevo y guárdalo como `dishonestcapacity.py`:

```
print(‘Enter TB or GB for the advertised unit:’)
```

```
unit = input(‘>’)
```

```
# Calcula la medida en la que miente la capacidad anunciada:
```

```

if unit == 'TB' or unit == 'tb':
    discrepancy = 1000000000000 / 1099511627776
elif unit == 'GB' or unit == 'gb':
    discrepancy = 1000000000 / 1073741824
print('Enter the advertised capacity:')
advertised_capacity = input('>')
advertised_capacity = float(advertised_capacity)
# Calcula la capacidad real, la redondea a la centena más próxima
# y la convierte en una cadena para poder concatenarla:
real_capacity = str(round(advertised_capacity * discrepancy, 2))
print('The actual capacity is ' + real_capacity + ' ' + unit)

```

Este programa pide al usuario que escriba la capacidad que dice tener el disco duro en TB o en GB:

```

# Calcula la medida en la que miente la capacidad anunciada:
if unit == 'TB' or unit == 'tb':
    discrepancy = 1000000000000 / 1099511627776
elif unit == 'GB' or unit == 'gb':
    discrepancy = 1000000000 / 1073741824

```

Los TB son más grandes que los GB y, cuanto más grande sea la unidad, mayor será la discrepancia entre las capacidades anunciada y real. Las declaraciones `if` y `elif` utilizan un operador booleano `or` para que el programa funcione al margen de si el usuario introduce la unidad en mayúscula o en minúscula. Si el usuario introduce otra cosa para la unidad, no se ejecutará ni la proposición `if` ni la proposición `elif`, y la variable `discrepancy` nunca se asignará. Después, cuando el programa intente usar esa variable, dará error.

Veremos ese caso luego.

A continuación, el usuario introduce el tamaño anunciado en las unidades dadas:

```
# Calcula la capacidad real, la redondea a la centena más próxima
```

```
# y la convierte en una cadena para poder concatenarla:
```

```
real_capacity = str(round(advertised_capacity * discrepancy, 2))
```

Pasan muchas cosas en esta sola línea. Vamos a usar el ejemplo del usuario introduciendo 10 TB como tamaño y unidad anunciados. Si nos fijamos en la parte más adentro del código, vemos que se multiplica `advertised_capacity` por `discrepancy`. Esa es la capacidad real, pero puede tener muchos dígitos, como en 9.094947017729282. Por eso, este número se pasa como primer argumento a `round()` con 2 como segundo argumento. Esta llamada a la función `round()` devuelve, en nuestro ejemplo, 9.09. Se trata de un valor flotante, pero queremos convertirlo en cadena para concatenarlo con un mensaje en la siguiente línea de código, para lo que se lo pasamos a la función `str()`. Python evalúa así esta línea de código:

```
real_capacity = str(round(advertised_capacity * discrepancy, 2))
                        ↓           ↓
real_capacity = str(round(10 * 0.9094947017729282, 2))
                        ↓
real_capacity = str(round(9.094947017729282, 2))
                        ↓
real_capacity = str(9.09)
                        ↓
real_capacity = '9.09'
```

Si el usuario no ha introducido TB, tb, GB or gb como unidad, las condiciones para las declaraciones `if` y `elif` serán `False` y nunca se creará la variable `discrepancy`. Pero el usuario no sabrá que hay algo mal hasta que Python intente usar la variable inexistente y muestre un error `NameError: name 'discrepancy' is not defined` y apunte a la línea en la que se ha asignado `real_capacity`.

Sin embargo, el verdadero origen de este error es el hecho que el programa no gestione la posibilidad de que el usuario introduzca

una unidad no válida. Hay muchas maneras de manejar este error, pero la más sencilla sería hacer que una proposición else muestre un mensaje como You must enter TB or GB y, a continuación, llame a la función sys.exit() para salir del programa. (Veremos más sobre esta función en el próximo capítulo).

La última línea del programa muestra la capacidad real del disco duro concatenando una cadena de mensaje con las cadenas real_capacity y unit strings:

```
print('The actual capacity is ' + real_capacity + ' ' + unit)
```

Por lo que se ve, los fabricantes de discos duros y memorias flash mienten más todavía: tengo una tarjeta SD de 256 GB en mi portátil que uso para copias de seguridad y en GB reales debería ser de 274.877.906.944 bytes y, en GB falsos, 256.000.000.000 bytes. Pero, según mi ordenador, la capacidad real es 255.802.212.352 bytes. Es curioso cómo el tamaño real es siempre inexacto por ser menor que el anunciado, nunca mayor.

Resumen

Con expresiones que se evalúan como True o False (también llamadas condiciones), podemos escribir programas que tomen decisiones sobre qué código ejecutar y qué código omitir. Estas condiciones son expresiones que comparan dos valores con los operadores ==, !=, <, >, <= y >= para evaluar como un valor booleano. También podemos usar los operadores booleanos and, or y not para conectar expresiones en expresiones más complejas. Python utiliza el sangrado para crear bloques de código. En este capítulo, hemos usado bloques como parte de declaraciones if, elif y else, pero, como iremos viendo, hay otras declaraciones de Python que también emplean bloques. Estas declaraciones de control de flujo te permitirán escribir programas más inteligentes.

Preguntas prácticas

1. ¿Cuáles son los dos valores del tipo de datos booleano? ¿Cómo se escriben?

2.¿Cuáles son los tres operadores booleanos?

3.Escribe la tabla de verdad para cada operador booleano, es decir, todas las combinaciones posibles de valores booleanos para cada operador y cómo se evalúan.

4.¿Cómo se evalúa la siguiente expresión?

$(5 > 4) \text{ and } (3 == 5)$

$\text{not } (5 > 4)$

$(5 > 4) \text{ or } (3 == 5)$

$\text{not } ((5 > 4) \text{ or } (3 == 5))$

$(\text{True and True}) \text{ and } (\text{True} == \text{False})$

$(\text{not False}) \text{ or } (\text{not True})$

5.¿Cuáles son los seis operadores de comparación?

6.¿Cuál es la diferencia entre el operador igual que y el operador de asignación?

7.Explica lo que es una condición y dónde usarías una.

8.Identifica los tres bloques de este código:

```
spam = 0
```

```
if spam == 10:
```

```
    print('eggs')
```

```
if spam > 5:
```

```
    print('bacon')
```

```
else:
```

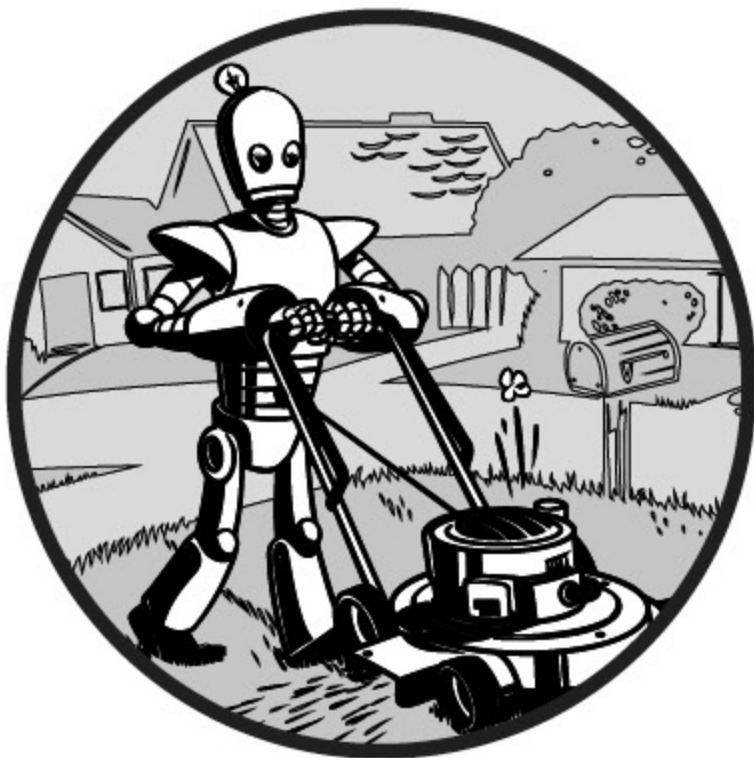
```
    print('ham')
```

```
print('spam')
```

```
print('Done')
```

9. Escribe un código que muestre Hello si spam almacena 1, Howdy si almacena 2 y Greetings! si almacena cualquier otra cosa.

Bucles



En el capítulo anterior, aprendiste a hacer que los programas ejecuten algunos bloques de código y se salten otros, pero el control de flujo va más allá. En este capítulo, aprenderás a ejecutar repetidamente bloques de código utilizando bucles. Los dos tipos de bucles de Python, `while` y `for`, desatan todo el poder de la automatización, ya que pueden ejecutar líneas de código millones de veces por segundo. También veremos cómo importar librerías de código, llamadas módulos, para poner más funciones todavía a disposición de tus programas.

Declaraciones de bucle `while`

Podemos hacer que un bloque de código se ejecute una y otra vez utilizando una declaración while. El código de la proposición while se ejecutará mientras la condición de la declaración sea True. El código de una declaración while siempre consta de lo siguiente:

- La palabra clave while.
- Una condición (es decir, una expresión que se evalúe como True o False).
- Dos puntos.
- A partir de la siguiente línea, un bloque de código sangrado (llamado proposición o bloque while).

Como verás, una declaración while se parece mucho a una declaración if. La diferencia está en cómo se comportan: al final de una proposición if, la ejecución del programa continúa después de la declaración if, mientras que, al final de una proposición while, la ejecución vuelve al principio de la declaración while. La proposición while se denomina a menudo el bucle while o, simplemente, el bucle.

Vamos a ver una declaración if y un bucle while que emplean la misma condición y realizan las mismas acciones basándose en dicha condición. Este es el código con una declaración if:

```
spam = 0
if spam < 5:
    print('Hello, world.')
    spam = spam + 1
```

Y este es el código con una declaración while:

```
spam = 0
while spam < 5:
    print('Hello, world.')
```

`spam = spam + 1`

Las dos declaraciones son similares: ambas comprueban el valor de `spam` y, si es menor que 5, muestran un mensaje. Pero, al ejecutar estos dos fragmentos de código, ocurrirá algo muy distinto con cada uno. Para la declaración `if`, la salida será sencillamente “Hello, world.”. En cambio, para la declaración `while`, será “Hello, world.” repetido cinco veces. Echa un vistazo a los diagramas de flujo de estos dos códigos, representados en las figuras 3.1 y 3.2, para ver por qué pasa eso.

El código con la declaración `if` comprueba la condición e imprime “Hello, world.” solo una vez si esa condición es verdadera. El código con el bucle `while`, por su parte, lo imprime cinco veces. El bucle se detiene después de cinco impresiones porque el entero de `spam` aumenta en uno al final de cada iteración del bucle, lo que significa que este se ejecutará cinco veces antes de que `spam < 5` sea `False`.

En el bucle `while`, la condición siempre se comprueba al inicio de cada iteración (es decir, cada vez que se ejecuta el bucle). Si la condición es `True`, se ejecuta la proposición y se vuelve a comprobar la condición. La primera vez que la condición resulte ser `False`, se omitirá el bucle `while`.

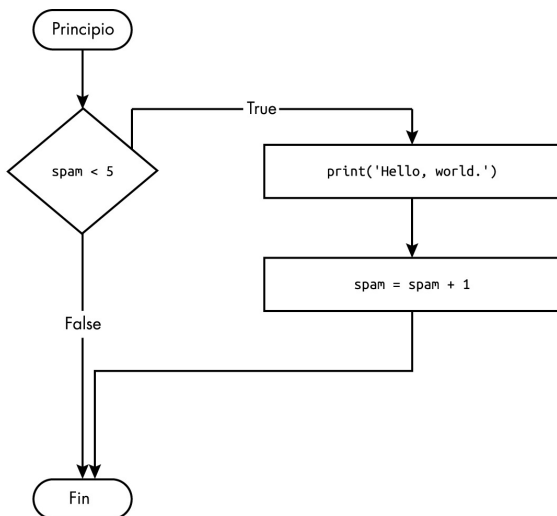


Figura 3.1. Diagrama de flujo para el código de la declaración if.

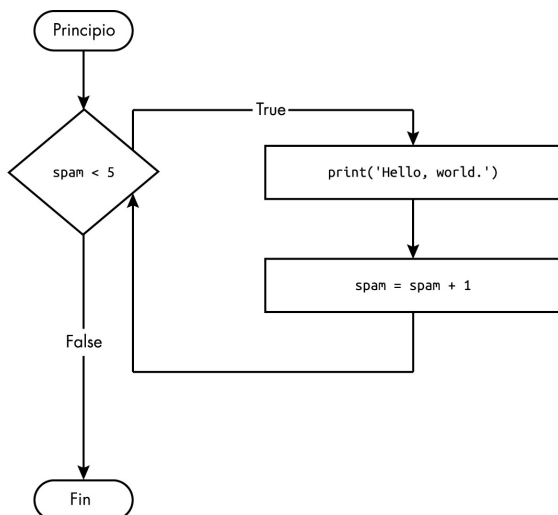


Figura 3.2. Diagrama de flujo para el código de la declaración while.

Un molesto bucle while

Vamos a ver un pequeño programa de ejemplo que estará todo el rato pidiéndonos escribir “your name” (no nuestro nombre, sino esas palabras exactas). Selecciona Archivo > Nuevo para abrir una ventana del editor de archivos, escribe el siguiente código y guárdalo como yourName.py:

```
name = ""

while name != 'your name':

    print('Please type your name.')

    name = input('>')

    print('Thank you!')
```

Primero, el programa establece la variable name en una cadena vacía. Lo hace para que la condición name != ‘your name’ se evalúe como True y la ejecución entre en la proposición del bucle

while.

El código de esta proposición pide al usuario que escriba su nombre, que se asigna a la variable `name`. Dado que es la última línea del código, la ejecución vuelve al principio del bucle para reevaluar la condición. Si el valor de `name` no es igual que la cadena `'your name'`, la condición es `True` y la ejecución vuelve a entrar en el bucle.

Pero, cuando el usuario deje de poner su nombre y escriba literalmente `your name`, la condición del bucle `while` será `'your name' != 'your name'`, que se evalúa como `False`. Ahora, en vez de volver a entrar en la proposición del bucle, Python la omite y sigue ejecutando el resto del programa. La figura 3.3 muestra el diagrama de flujo para el programa `yourName.py`.

Vamos a ver `yourName.py` en acción. Pulsa **F5** para ejecutarlo y escribe algo que no sea `your name` unas cuantas veces antes de dar al programa lo que quiere:

Please type your name.

> **Al**

Please type your name.

> **Albert**

Please type your name.

> **%#@#%*(^&!!!**

Please type your name.

> **your name**

Thank you!

Si nunca escribes `your name`, la condición del bucle `while` nunca será `False` y el programa seguirá haciendo la misma petición para siempre. Aquí, la llamada a `input()` permite al usuario introducir la cadena correcta para que el programa avance. En otros programas,

la condición podría no cambiar nunca, lo cual puede ser un problema. Veamos cómo se puede escapar de un bucle while.

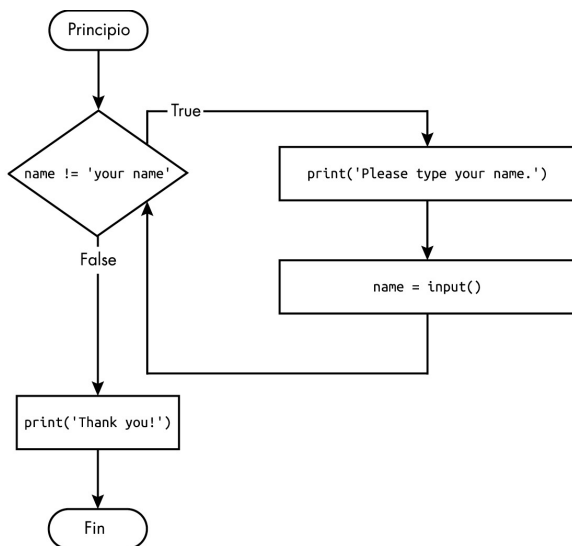


Figura 3.3. Diagrama de flujo para el programa yourName.py.

Declaraciones break

Hay un atajo para hacer que la ejecución del programa salga de la proposición de un bucle while prematuramente. Si la ejecución llega a una declaración break, sale inmediatamente de la proposición del bucle while. En código, una declaración break contiene simplemente la palabra clave break.

El siguiente programa hace lo mismo que el programa yourName.py anterior, pero utiliza una declaración break para salir del bucle. Escribe el siguiente código y guarda el archivo como yourName2.py:

```
while True:
```

```
    print('Please type your name.')
```

```
    name = input('>')
```

```
    if name == 'your name':
```

```
break
```

```
print("Thank you!")
```

La primera línea crea un bucle infinito; es un bucle while cuya condición es siempre True. (Después de todo, la expresión True siempre se evalúa al valor True). Una vez que la ejecución del programa entre en este bucle, solo saldrá cuando se ejecute una declaración break. (Un bucle infinito del que no se puede salir es un error de programación habitual).

Igual que antes, este programa pide al usuario que escriba your name. Sin embargo, ahora, mientras la ejecución está todavía dentro del bucle while, una declaración if comprueba si name es igual que 'your name'. Si esta condición es True, se ejecuta la declaración break y la ejecución sale del bucle y avanza hasta print("Thank you!"). Si no, la declaración if que contiene la declaración break se omite, lo que coloca la ejecución al final del bucle while. En este punto, la ejecución del programa salta de nuevo al principio de la declaración while para volver a comprobar la condición. Dado que esta condición es el valor booleano True, sin más, la ejecución entra en el bucle para pedir al usuario de nuevo que introduzca your name. La figura 3.4 muestra el diagrama de flujo para este programa.

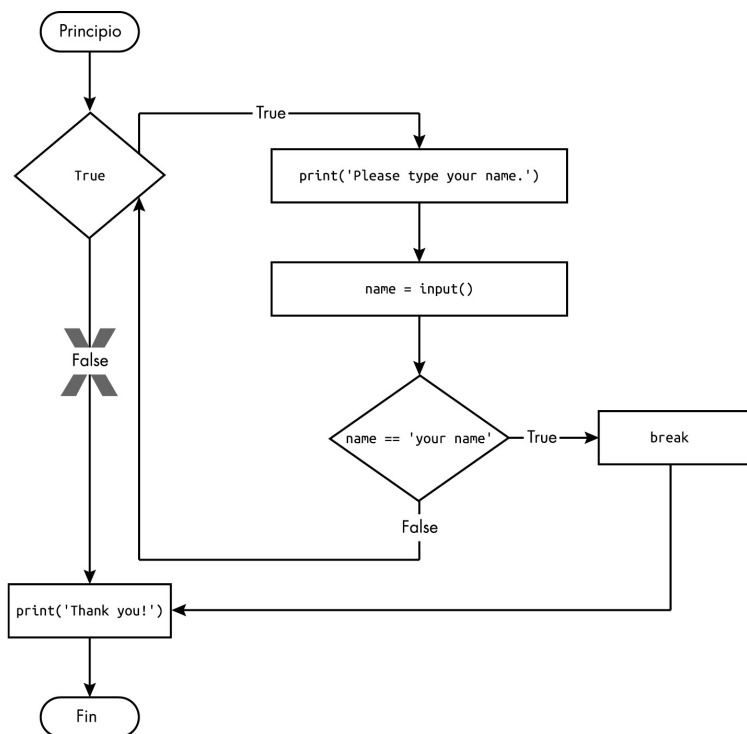


Figura 3.4. Diagrama de flujo para el programa yourName2.py con un bucle infinito. Observa que la ruta X nunca sucederá de manera lógica porque la condición del bucle es siempre True.

Ejecuta yourName2.py e introduce el mismo texto que para yourName.py. El programa reescrito debería responder igual que el original.

Declaraciones continue

Al igual que sucede con las declaraciones break, podemos utilizar declaraciones continue dentro de bucles. Cuando la ejecución del programa llega a una declaración continue, la ejecución del programa vuelve inmediatamente al inicio del bucle y reevalúa la condición. (Esto también ocurre cuando la ejecución llega al final del bucle).

¿Atrapado en un bucle infinito?

Si alguna vez ejecutas un programa que tiene un error que hace que quedes atrapado en un bucle infinito, pulsa **Control-C**. Esto enviará un error KeyboardInterrupt al programa y lo detendrá de inmediato. También se puede hacer clic en el botón Detener de la parte superior de la ventana de Mu. Prueba a detener un programa creando un bucle infinito en el editor de archivos y guarda el programa como infiniteLoop.py. Si estás ejecutando el programa desde Mu, también puedes hacer clic en Detener:

```
while True:
```

```
    print('Hello, world!')
```

Cuando ejecutes este programa, imprimirá Hello, world! en la pantalla para siempre porque la condición del bucle while es siempre True. **Control-C** también es útil si quieres salir del programa inmediatamente, aunque no estés en un bucle infinito.

Vamos usar continue para escribir un programa que pida un nombre y una contraseña. Escribe el siguiente código en una nueva ventana del editor de archivos y guarda el programa como swordfish.py:

```
while True:
```

```
    print('Who are you?')
```

```
    name = input('>')
```

```
    1 if name != 'Joe':
```

```
        2 continue
```

```
    print('Hello, Joe. What is the password? (It is a fish.)')
```

```
    3 password = input('>')
```

```
    if password == 'swordfish':
```

```
        4 break
```

```
    5 print('Access granted.')
```


Si el usuario introduce cualquier nombre distinto de Joe 1, la declaración continue 2 hace que la ejecución del programa vuelva al principio del bucle. Cuando el programa reevalúa la condición, la ejecución entrará siempre en el bucle porque es sencillamente el valor True. Cuando el usuario consigue pasar de esa declaración if, se le pide una contraseña 3. Si la contraseña introducida es swordfish, la declaración break 4 se ejecuta y la ejecución sale del bucle while para mostrar Access granted. 5 De lo contrario, la ejecución continúa hasta el final del bucle while, donde después salta de nuevo al inicio del bucle. Mira la figura 3.5 para ver el diagrama de flujo de este programa.

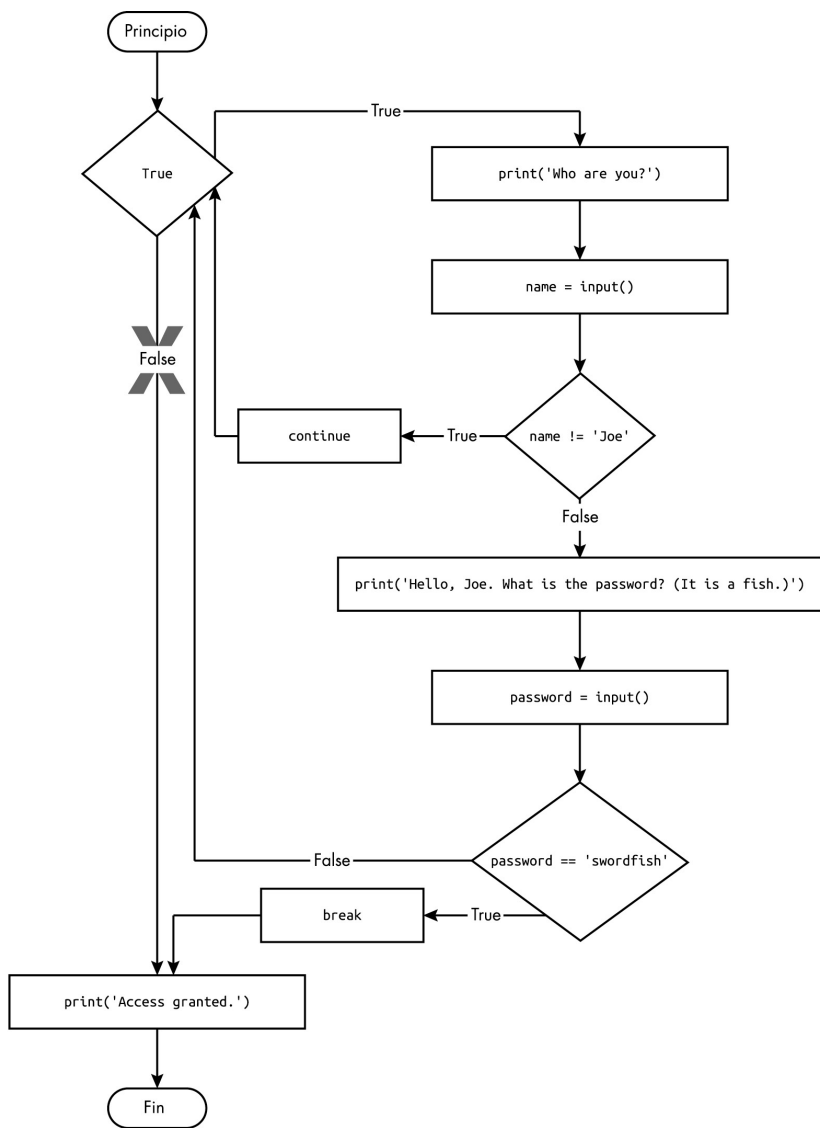


Figura 3.5. Diagrama de flujo para el programa swordfish.py. La ruta X nunca se producirá lógicamente porque la condición del bucle es siempre True.

Ejecuta este programa e introduce algún dato. Mientras no digas que eres Joe, el programa no debería pedirte una contraseña y, cuando escribas la contraseña correcta, debería salir:

Who are you?

> **I'm fine, thanks. Who are you?**

Who are you?

> **Joe**

Hello, Joe. What is the password? (It is a fish.)

> **Mary**

Who are you?

> **Joe**

Hello, Joe. What is the password? (It is a fish.)

> **swordfish**

Access granted.

Valores truthy y falsey y la función bool()

Las condiciones considerarán algunos valores en otros tipos de datos equivalentes a True o False. Cuando se usan en condiciones, 0, 0.0 y '' (la cadena vacía) se consideran False, mientras que todos los demás valores se consideran True. Por ejemplo, observa el siguiente programa:

```
name = ''
```

```
1 while not name:
```

```
    print('Enter your name:')
```

```
    name = input('>')
```

```
    print('How many guests will you have?')
```

```
    num_of_guests = int(input('>'))
```

```
2 if num_of_guests:
```

```
3 print('Be sure to have enough room for all your guests.')
```

```
print('Done')
```

Si el usuario introduce una cadena vacía para `name`, la condición de la declaración `while` será `True` 1, y el programa seguirá pidiendo un nombre. Si el valor para `num_of_guests` no es 0 2, la condición se considera `True` y el programa imprimirá un recordatorio para el usuario 3.

Podríamos haber escrito `not name != ''` en vez de `not name` y `num_of_guests != 0` en vez de `num_of_guests`, pero usar los valores *truthy* y *falsey* puede hacer que resulte más fácil leer el código.

Si quieres saber si un valor es *truthy* o *falsey*, pásaselo a la función `bool()` como en este ejemplo con el intérprete de comandos interactivo:

```
> > > bool(0)
```

False

```
> > > bool(42)
```

True

```
> > > bool('Hello')
```

True

```
> > > bool('')
```

False

Bucles `for` y la función `range()`

El bucle `while` seguirá ejecutándose mientras su condición sea `True`, pero ¿y si queremos ejecutar un bloque de código solo un número determinado de veces? Podemos hacerlo con una declaración de bucle `for` y la función `range()`.

En código, una declaración `for` tiene un aspecto parecido a `for i in`

range(5): e incluye lo siguiente:

- La palabra clave for.
- Un nombre de variable.
- La palabra clave in.
- Una llamada a la función range() con hasta tres enteros que se le pasen.
- Dos puntos.
- A partir de la siguiente línea, un bloque de código sangrado (denominado proposición o bloque for).

Vamos a crear un programa nuevo llamado fiveTimes.py para que veas el bucle for en acción:

```
print('Hello!')  
  
for i in range(5):  
  
    print("On this iteration, i is set to " + str(i))  
  
    print("Goodbye!")
```

El código de la proposición del bucle for se ejecuta cinco veces. La primera, la variable i se establece en 0. La llamada a print() en la proposición imprimirá On this iteration, i is set to 0. Cuando Python termine una iteración por todo el código dentro de la proposición for, la ejecución vuelve al principio del bucle y la declaración for incrementa i en uno. Por eso, el resultado de range(5) son cinco iteraciones por la proposición, con i estableciéndose como 0, luego 1, 2, 3 y 4. La variable i ascenderá al entero pasado a range(), aunque no lo incluirá. La figura 3.6 muestra el diagrama de flujo para el programa fiveTimes.py.

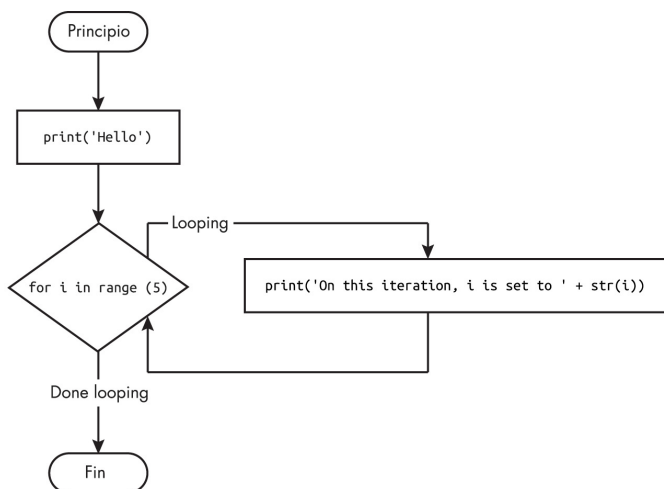


Figura 3.6. Diagrama de flujo para el programa fiveTimes.py.

Esta es la salida completa del programa:

Hello!

On this iteration, i is set to 0

On this iteration, i is set to 1

On this iteration, i is set to 2

On this iteration, i is set to 3

On this iteration, i is set to 4

Goodbye!

También podemos utilizar declaraciones `break` y `continue` dentro de bucles `for`. La declaración `continue` continuará hasta el siguiente valor del contador del bucle, como si la ejecución del programa hubiese llegado al final del bucle y regresado al principio. De hecho, podemos usar declaraciones `continue` y `break` solo dentro de bucles `while` y `for`. Si intentamos utilizar estas declaraciones en cualquier otra parte, Python dará error.

Otro ejemplo para el bucle `for`: piensa en la siguiente historia sobre

el matemático Carl Friedrich Gauss. Cuando Gauss era pequeño, un profesor quiso poner a la clase un trabajo de relleno. Les dijo que sumasen todos los números de 0 a 100. Al joven Gauss se le ocurrió un truco para averiguar el resultado en unos segundos, pero nosotros podemos escribir un programa de Python con un bucle for para que nos haga el cálculo:

```
total = 0

for num in range(101):

    total = total + num

print(total)
```

El resultado debería ser 5.050. Cuando el programa empieza, la variable total está a 0. Después, el bucle for ejecuta $total = total + num$ 101 veces, cada vez con una variable num incrementada. Para cuando el bucle termina las 101 iteraciones, cada entero de 0 a 100 se ha sumado al total. En ese punto, se imprime total en la pantalla. Incluso en los ordenadores más lentos, este programa tarda menos de un segundo en completarse.

(El joven Gauss descubrió una forma de resolver el problema en segundos. Hay 50 pares de números que sumar hasta 101: $1 + 100$, $2 + 99$, $3 + 98$, etc., hasta $50 + 51$. Dado que 50×101 es 5050, la suma de todos los números de 0 a 100 es 5050. ¡Chico listo!).

¿Por qué “hasta pero no incluido”?

Puede parecer extraño que la función range() del bucle for especifique el número hasta el que llega el bucle, menos uno. En programación, los rangos se especifican habitualmente en un formato “cerrado, abierto” que incluye el número inicial, pero excluye el número final. Por ejemplo, el rango 0, 10 incluiría los números del 0 al 9 en lugar del 0 al 10. Usar el formato “cerrado, abierto” tiene muchas ventajas sobre el “cerrado, cerrado”, que conduce a unos cuantos fallos. Calcular el tamaño del rango es solo cuestión de restar el número inicial al final. El rango 0, 10 (que contiene los números del 0 al 9 y no al 10) tiene $10 - 0$ o 10 números. Con un rango “cerrado, cerrado” de 0, 9, hay que calcular

la longitud como $9 - 0 + 1$. (Y es fácil cometer errores de quitar uno como $9 - 0 - 1$). El inicio del siguiente rango 10, 20 es solo cuestión de usar el número final anterior como nuevo número de inicio.

A los programas que manejan sellos de tiempo les resulta mucho más fácil trabajar con el rango “cerrado, abierto” de 00:00:00, 24:00:00,0 que con el rango “cerrado, cerrado” de 00:00:00, 23:59:59,999. Los rangos “cerrado, abierto” pueden parecer raros al principio, pero enseguida te resultarán naturales a medida que cojas experiencia escribiendo código.

Un bucle while equivalente

Podemos usar un bucle while para que haga lo mismo que un bucle for; los bucles for son simplemente más concisos. Vamos a reescribir fiveTimes.py para utilizar un bucle while equivalente de un bucle for:

```
print('Hello!')
```

```
i = 0
```

```
while i < 5:
```

```
    print('On this iteration, i is set to ' + str(i))
```

```
    i = i + 1
```

```
print('Goodbye!')
```

Si ejecutamos este programa, la salida debería ser la misma que la del programa fiveTimes.py que usaba el bucle for. Recuerda que los bucles for son útiles para pasar por el bucle un número específico de veces y los while para pasar mientras se dé una condición determinada.

Argumentos para range()

Algunas funciones pueden llamarse con múltiples argumentos separados por una coma, y range() es una de ellas. Esto nos permite

cambiar el entero que pasa a range() para seguir cualquier secuencia de enteros, lo que puede incluir un número inicial distinto de cero:

```
for i in range(12, 16):
```

```
    print(i)
```

El primer argumento será donde comience la variable del bucle for y el segundo será hasta el número donde se detendrá, sin incluirlo:

12

13

14

15

La función range() también puede llamarse con tres argumentos. Los dos primeros serán los valores de inicio y fin y el tercero, el argumento de paso. El paso es la cantidad en la que se incrementa la variable tras cada iteración:

```
for i in range(0, 10, 2):
```

```
    print(i)
```

Así pues, llamar a range(0, 10, 2) contará de cero a ocho en intervalos de dos:

0

2

4

6

8

La función range() es flexible en la secuencia de números que produce para los bucles for. Por ejemplo, podemos incluso utilizar

un número negativo como argumento de paso para hacer que el bucle cuente hacia atrás en vez de hacia delante:

```
for i in range(5, -1, -1):
```

```
    print(i)
```

Este bucle for tendría la siguiente salida:

5

4

3

2

1

0

Ejecutar un bucle for para imprimir i con range(5, -1, -1) debería imprimir los números de cinco a cero.

Importación de módulos

Todos los programas de Python pueden llamar a un conjunto básico de funciones denominadas funciones integradas, incluidas las funciones print(), input() y len() que ya hemos visto. Python también incluye un conjunto de módulos llamado la biblioteca estándar. Cada módulo es un programa de Python que contiene un grupo relacionado de funciones que podemos incrustar en nuestros programas. Por ejemplo, el módulo math tiene funciones relacionadas con las matemáticas, el módulo random funciones relacionadas con números aleatorios, etc.

Para poder usar las funciones de un módulo, antes hay que importarlo con una declaración import. En código, esta declaración consta de lo siguiente:

- La palabra clave import.

- El nombre del módulo.
- Opcionalmente, más nombres de módulos, siempre que estén separados por comas.

Una vez importado un módulo, ya podemos usar todas sus funciones. Vamos a probar con el módulo `random`, que nos dará acceso a la función `random.randint()`.

Escribe este código en el editor de archivos y guárdalo como `printRandom.py`:

```
import random  
  
for i in range(5):  
  
    print(random.randint(1, 10))
```

Al ejecutar este programa, la salida será algo así:

```
4  
  
1  
  
8  
  
4  
  
1
```

La llamada a la función `random.randint()` se evalúa a un valor entero aleatorio entre los dos enteros que le pasamos. Como `randint()` está en el módulo `random`, primero hay que escribir `random.` delante del nombre de la función para decir a Python que busque la función dentro de ese módulo.

No sobrescribas los nombres de módulos

Cuando guardes tus *scripts* de Python, ten cuidado de no ponerles un nombre de los que utilizan los nombres de los módulos de Python, como `random.py`, `sys.py`, `os.py` o `math.py`. Si llamas a uno de tus programas por accidente `random.py`, por ejemplo, y usas una

declaración `import random` en otro programa, este importará tu archivo `random.py` en lugar del módulo `random` de Python. Esto puede llevar a errores como `AttributeError: module 'random' has no attribute 'randint'`, ya que tu archivo `random.py` no tiene las funciones del verdadero módulo `random`. Tampoco utilices los nombres de las funciones integradas de Python para tus archivos ni nombres de variables. Algunos nombres típicos de Python incluyen `all`, `any`, `date`, `email`, `file`, `format`, `hash`, `id`, `input`, `list`, `min`, `max`, `object`, `open`, `random`, `set`, `str`, `sum`, `test` y `type`.

Aquí tenemos un ejemplo de declaración `import` que importa cuatro módulos diferentes:

```
import random, sys, os, math
```

Ahora podemos usar cualquiera de las funciones de estos cuatro módulos. Volveremos sobre ellos más adelante en el libro.

Hay una forma alternativa de la declaración `import` que consiste en la palabra clave `from`, seguida del nombre del módulo, la palabra clave `import` y un asterisco (*); por ejemplo, `from random import *`. Con esta forma de declaración `import`, las llamadas a funciones en `random` no necesitarán el prefijo `random..` No obstante, utilizar el nombre completo hace el código más legible, así que es mejor usar la forma `import random`.

Terminar un programa antes de tiempo con `sys.exit()`

El último concepto de control de flujo que vamos a tratar es cómo terminar el programa o salir de él. Los programas siempre terminan si la ejecución llega al final de las instrucciones. Sin embargo, podemos hacer que el programa termine antes de la última instrucción llamando a la función `sys.exit()`. Dado que se encuentra en el módulo `sys`, habrá que importar este módulo para que el programa pueda usarlo.

Abre una ventana del editor de archivos y escribe el siguiente código, guardándolo como `exitExample.py`:

```
import sys
```

while True:

```
print("Type exit to exit.")
```

```
response = input('>')
```

```
if response == 'exit':
```

```
sys.exit()
```

```
print("You typed " + response + ".")
```

Ejecuta este programa en tu editor de código. Tiene un bucle infinito sin declaración break. La única forma de que el programa termine es que la ejecución llegue a la llamada a `sys.exit()`. Cuando `response` es igual que `'exit'`, se ejecuta la línea que contiene la llamada a `sys.exit()`. Dado que la variable `response` está establecida por la función `input()`, el usuario debe escribir **exit** para detener el programa.

Un programa corto: adivina el número

Los ejemplos que te he mostrado hasta ahora son útiles para presentar conceptos básicos, pero ahora verás cómo aplicar todo lo que has aprendido en un programa más completo. En este apartado, te mostraré un sencillo juego de adivinación de números. Al ejecutar este programa, la salida será algo así:

I am thinking of a number between 1 and 20.

Take a guess.

> 10

Your guess is too low.

Take a guess.

> 15

Your guess is too low.

Take a guess.

> **17**

Your guess is too high.

Take a guess.

> **16**

Good job! You got it in 4 guesses!

Introduce el siguiente código fuente en el editor de archivos y guarda como guessTheNumber.py:

```
# Esto es un juego de adivinar números.
```

```
import random
```

```
secret_number = random.randint(1, 20)
```

```
print('I am thinking of a number between 1 and 20.')
```

```
# Pide al jugador que adivine 6 veces.
```

```
for guesses_taken in range(1, 7):
```

```
    print('Take a guess.')
```

```
    guess = int(input('> '))
```

```
    if guess < secret_number:
```

```
        print('Your guess is too low.')
```

```
    elif guess > secret_number:
```

```
        print('Your guess is too high.')
```

```
    else:
```

```
        break # ¡Esta condición es la buena!
```

```
if guess == secret_number:
```

```
    print("Good job! You got it in " + str(guesses_taken) + " guesses!")
```

```
else:
```

```
    print("Nope. The number was " + str(secret_number))
```

Echemos un vistazo al código, línea por línea, desde el principio:

```
# Esto es un juego de adivinar números.
```

```
import random
```

```
secret_number = random.randint(1, 20)
```

Primero, un comentario al principio del código explica lo que hace el programa. A continuación, el programa importa el módulo `random` para poder usar la función `random.randint()` para generar el número que tiene que adivinar el usuario. El valor devuelto, un entero aleatorio entre 1 y 20, se almacena en la variable `secret_number`:

```
print('I am thinking of a number between 1 and 20.')
```

```
# Pide al jugador que adivine 6 veces.
```

```
for guesses_taken in range(1, 7):
```

```
    print("Take a guess.")
```

```
    guess = int(input('> '))
```

El programa dice al jugador que tiene un número secreto y le da seis oportunidades para averiguarlo. El código deja al jugador escribir un número y lo comprueba en un bucle `for` como máximo seis veces. Lo primero que ocurre en el bucle es que el jugador escribe su intento. Como `input()` devuelve una cadena, el valor devuelto pasa directamente a `int()`, que convierte la cadena en un valor entero, que se almacena en la variable `guess`:

```
if guess < secret_number:
```

```
print('Your guess is too low.')
```

```
elif guess > secret_number:
```

```
print('Your guess is too high.')
```

Estas pocas líneas de código comprueban si el número es menor o mayor que el número secreto. En cada caso, aparece una pista en la pantalla:

```
else:
```

```
break # ¡Esta condición es la buena!
```

Si el número del jugador no es ni mayor ni menor que el secreto, debe de ser igual, en cuyo caso queremos que la ejecución del programa salga del bucle for:

```
if guess == secret_number:
```

```
print('Good job! You got it in ' + str(guesses_taken) + ' guesses!')
```

```
else:
```

```
print('The number was ' + str(secret_number))
```

Después del bucle for, la anterior declaración if-else comprueba si el jugador ha adivinado el número e imprime el mensaje correspondiente en la pantalla. En ambos casos, el programa muestra una variable que contiene un valor entero (guesses_taken y secret_number). Puesto que debe concatenar estos valores enteros en cadenas, pasa las variables a la función str(), que devuelve el valor de cadena de esos enteros. Ahora, se puede concatenar estas cadenas con el operador + antes de pasarlas finalmente a la llamada a la función print().

Un programa corto: piedra, papel o tijera

Vamos a emplear los conceptos de programación que hemos aprendido hasta ahora para crear un sencillo juego de piedra, papel o tijera, cuya salida tendrá este aspecto:

ROCK, PAPER, SCISSORS

0 Wins, 0 Losses, 0 Ties

Enter your move: (r)ock (p)aper (s)cissors or (q)uit

> **p**

PAPER versus...

PAPER

It is a tie!

0 Wins, 0 Losses, 1 Ties

Enter your move: (r)ock (p)aper (s)cissors or (q)uit

> **s**

SCISSORS versus...

PAPER

You win!

1 Wins, 0 Losses, 1 Ties

Enter your move: (r)ock (p)aper (s)cissors or (q)uit

> **q**

Escribe el siguiente código fuente en el editor de archivos y guarda el fichero como rpsGame.py:

```
import random, sys
```

```
print('ROCK, PAPER, SCISSORS')
```

```
# Estas variables llevan la cuenta de victorias, derrotas y empates.
```

```
wins = 0
```

```
losses = 0

ties = 0

while True: # El bucle principal del juego.

    print('%s Wins, %s Losses, %s Ties' % (wins, losses, ties))

    while True: # El bucle de entrada del jugador

        print('Enter your move: (r)ock (p)aper (s)cissors or (q)uit')

        player_move = input('>')

        if player_move == 'q':

            sys.exit() # Sale del programa.

        if player_move == 'r' or player_move == 'p' or player_move == 's':

            break # Sale del bucle de entrada del jugador.

        print('Type one of r, p, s, or q.')

    # Muestra lo que elige el jugador:

    if player_move == 'r':

        print('ROCK versus...')

    elif player_move == 'p':

        print('PAPER versus...')

    elif player_move == 's':

        print('SCISSORS versus...')

    # Muestra lo que elige el ordenador:

    move_number = random.randint(1, 3)
```

```
if move_number == 1:
    computer_move = 'r'
    print('ROCK')
elif move_number == 2:
    computer_move = 'p'
    print('PAPER')
elif move_number == 3:
    computer_move = 's'
    print('SCISSORS')
# Muestra el registro de victorias/derrotas/empates:
if player_move == computer_move:
    print('It is a tie!')
    ties = ties + 1
elif player_move == 'r' and computer_move == 's':
    print('You win!')
    wins = wins + 1
elif player_move == 'p' and computer_move == 'r':
    print('You win!')
    wins = wins + 1
elif player_move == 's' and computer_move == 'p':
    print('You win!')
    wins = wins + 1
```

```
elif player_move == 'r' and computer_move == 'p':
```

```
    print("You lose!")
```

```
    losses = losses + 1
```

```
elif player_move == 'p' and computer_move == 's':
```

```
    print("You lose!")
```

```
    losses = losses + 1
```

```
elif player_move == 's' and computer_move == 'r':
```

```
    print("You lose!")
```

```
    losses = losses + 1
```

Veamos el código, línea por línea, desde el principio:

```
import random, sys
```

```
print('ROCK, PAPER, SCISSORS')
```

```
# Estas variables llevan la cuenta de victorias, derrotas y empates.
```

```
wins = 0
```

```
losses = 0
```

```
ties = 0
```

Primero, importamos los módulos random y sys para que nuestro programa pueda llamar a las funciones random.randint() y sys.exit(). También usamos tres variables para llevar la cuenta de victorias, derrotas y empates del jugador:

```
while True: # El bucle principal del juego.
```

```
    print('%s Wins, %s Losses, %s Ties' % (wins, losses, ties))
```

```
    while True: # El bucle de entrada del jugador
```

```

print('Enter your move: (r)ock (p)aper (s)cissors or (q)uit')

player_move = input('>')

if player_move == 'q':

    sys.exit() # Sale del programa.

if player_move == 'r' or player_move == 'p' or player_move == 's':

    break # Sale del bucle de entrada del jugador.

print('Type one of r, p, s, or q.')
```

Este programa utiliza un bucle while dentro de otro bucle while. El primero es el bucle principal del juego y se juega un nuevo juego de piedra, papel o tijera en cada iteración por este bucle. El segundo bucle pide entrada al jugador y sigue pasando en bucle hasta que el jugador introduce r, p, s o q para su movimiento. La r, la p y la s se corresponden con piedra (*rock*), papel (*paper*) y tijera (*scissors*), respectivamente, mientras que la q significa que el jugador quiere dejarlo (*quit*). En ese caso, se llama a `sys.exit()` y el programa termina. Si el jugador introduce r, p o s, la ejecución sale del bucle. De lo contrario, el programa recuerda al jugador que debe escribir **r, p, s o q** y vuelve al principio del bucle:

Muestra lo que elige el jugador:

```

if player_move == 'r':

    print('ROCK versus...')

elif player_move == 'p':

    print('PAPER versus...')

elif player_move == 's':

    print('SCISSORS versus...')
```

El movimiento del jugador se muestra en la pantalla:

```
# Muestra lo que elige el ordenador:

move_number = random.randint(1, 3)

if move_number == 1:

    computer_move = 'r'

    print('ROCK')

elif move_number == 2:

    computer_move = 'p'

    print('PAPER')

elif move_number == 3:

    computer_move = 's'

    print('SCISSORS')
```

A continuación, el programa selecciona aleatoriamente el movimiento del ordenador. Como `random.randint()` siempre devuelve un número aleatorio, el código guarda el valor entero 1, 2 o 3 en una variable llamada `move_number`. Después, el programa almacena una cadena 'r', 'p' o 's' en `computer_move` basándose en el entero de `move_number`, además de mostrar el movimiento del ordenador:

```
# Muestra el registro de victorias/derrotas/empates:

if player_move == computer_move:

    print('It is a tie!')

    ties = ties + 1

elif player_move == 'r' and computer_move == 's':

    print('You win!')

    wins = wins + 1
```

```
elif player_move == 'p' and computer_move == 'r':  
    print("You win!")  
    wins = wins + 1  
elif player_move == 's' and computer_move == 'p':  
    print("You win!")  
    wins = wins + 1  
elif player_move == 'r' and computer_move == 'p':  
    print("You lose!")  
    losses = losses + 1  
elif player_move == 'p' and computer_move == 's':  
    print("You lose!")  
    losses = losses + 1  
elif player_move == 's' and computer_move == 'r':  
    print("You lose!")  
    losses = losses + 1
```

Por último, el programa compara las cadenas de `player_move` y `computer_move` y muestra el resultado en la pantalla. También aumenta la variable `wins`, `losses` o `ties` en consecuencia. Cuando la ejecución llega al final, vuelve al principio del bucle principal para iniciar otro juego.

Si te han gustado estos juegos de adivinar números y piedra, papel o tijera, encontrarás el código fuente de otros programas sencillos de Python en *The Big Book of Small Python Projects* (No Starch Press, 2021).

Resumen

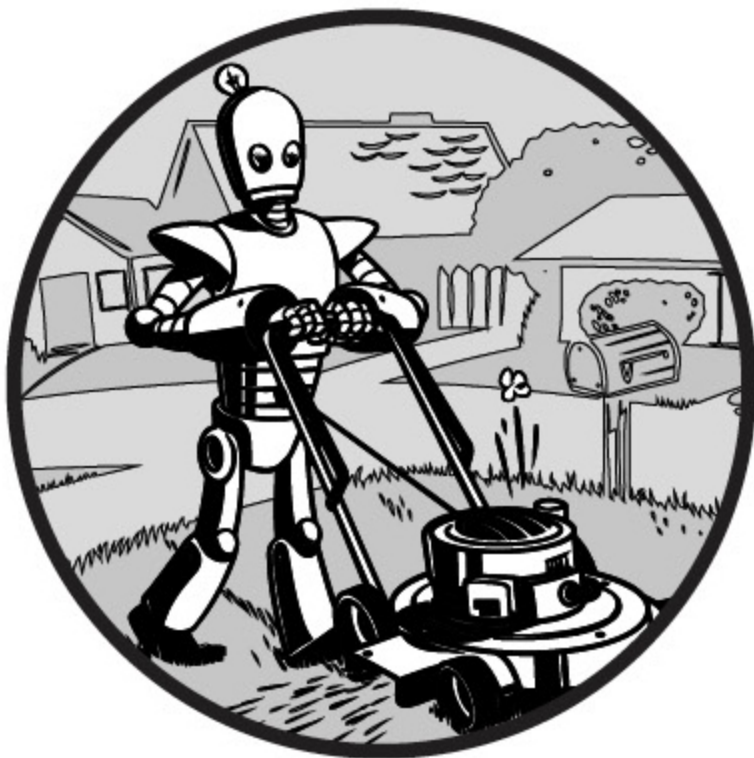
Los bucles permiten ejecutar código una y otra vez mientras una condición determinada se evalúe como True. Las declaraciones `break` y `continue` son útiles si necesitas salir de un bucle o saltar al principio.

Estas declaraciones de control de flujo te permitirán escribir programas más inteligentes. También puedes usar otro tipo de control de flujo escribiendo tus propias funciones, que es el tema del próximo capítulo.

Preguntas prácticas

1. ¿Qué teclas puedes pulsar si tu programa de Python se atasca en un bucle infinito?
2. ¿Cuál es la diferencia entre `break` y `continue`?
3. ¿Cuál es la diferencia entre `range(10)`, `range(0, 10)` y `range(0, 10, 1)` en un bucle `for`?
4. Escribe un programa corto que imprima los números del 1 al 10 con un bucle `for`. Después, escribe un programa equivalente que imprima los números del 1 al 10 con un bucle `while`.
5. Si tuvieses una función llamada `bacon()` dentro de un módulo llamado `spam`, ¿cómo la llamarías después de importar `spam`?

Funciones



Una función es como un miniprograma dentro de un programa. Python ofrece una serie de funciones integradas, las funciones `print()`, `input()` y `len()` de los capítulos anteriores, pero también podemos escribir funciones propias. En este capítulo, crearás funciones, explorarás la pila de llamadas que sirve para determinar el orden en el que se ejecutan las funciones de un programa y aprenderás sobre el ámbito de las variables dentro y fuera de las funciones.

Crear funciones

Para entender mejor cómo funcionan las funciones, vamos a crear una. Escribe este programa en el editor de archivos y guárdalo como `helloFunc.py`:

```
def hello():  
  
    # Imprime tres saludos  
  
    print('Good morning!')  
  
    print('Good afternoon!')  
  
    print('Good evening!')  
  
    hello()  
  
    hello()  
  
    print('ONE MORE TIME!')  
  
    hello()
```

La primera línea es una declaración `def`, que define una función llamada `hello()`. El código del bloque que sigue a la declaración `def` es el cuerpo de la función. Este código se ejecuta cuando se llama a la función, no cuando se define la misma.

Las líneas `hello()` después de la función son llamadas a la función. En código, una llamada a una función es simplemente el nombre de la función seguido de unos paréntesis, tal vez con algunos argumentos entre los paréntesis. Cuando la ejecución del programa llega a estas llamadas, salta a la primera línea de la función y empieza a ejecutar ese código. Cuando llega al final de la función, la ejecución vuelve a la línea que llamaba a la función y sigue avanzando por el código como antes.

Como este programa llama a `hello()` tres veces, el código de la función se ejecuta tres veces. Al ejecutar el programa, la salida será así:

Good morning!

Good afternoon!

Good evening!

Good morning!

Good afternoon!

Good evening!

ONE MORE TIME!

Good morning!

Good afternoon!

Good evening!

Uno de los principales fines de las funciones es agrupar código que se ejecuta varias veces. Sin una función definida, habría que copiar y pegar este código cada vez que quisiéramos ejecutarlo y el programa quedaría así:

```
print("Good morning!")
```

```
print("Good afternoon!")
```

```
print("Good evening!")
```

```
print("Good morning!")
```

```
print("Good afternoon!")
```

```
print("Good evening!")
```

```
print("ONE MORE TIME!")
```

```
print("Good morning!")
```

```
print("Good afternoon!")
```

```
print(' Good evening!')
```

En general, conviene siempre evitar duplicar código porque, si alguna vez hay que actualizarlo (por ejemplo, porque descubrimos un fallo que hay que corregir), tendremos que recordar cambiar el código en todas partes donde lo hayamos copiado.

A medida que cojas experiencia programando, te darás cuenta de que te vas deshaciendo de código copiado y pegado. Esto hace que los programas sean más cortos, más fáciles de leer y más fáciles de actualizar.

Argumentos y parámetros

Cuando llamamos a la función `print()` o `len()`, le pasamos valores, llamados argumentos, poniéndolos entre los paréntesis. También podemos definir nuestras propias funciones que acepten argumentos. Escribe este ejemplo en el editor de archivos y guárdalo como `helloFunc2.py`:

```
1 def say_hello_to(name):  
  
# Imprime tres saludos para el nombre proporcionado.  
  
2 print('Good morning, ' + name)  
  
print('Good afternoon, ' + name)  
  
print('Good evening, ' + name)  
  
3 say_hello_to('Alice')  
  
say_hello_to('Bob')
```

Al ejecutar el programa, la salida será así:

Good morning, Alice

Good afternoon, Alice

Good evening, Alice

Good morning, Bob

Good afternoon, Bob

Good evening, Bob

La definición de la función `say_hello_to()` tiene un parámetro llamado `name` 1. Los parámetros son variables que contienen argumentos. Cuando se llama a una función con argumentos, estos se almacenan en los parámetros. La primera vez que se llama a la función `say_hello_to()`, se le pasa el argumento 'Alice' 3. La ejecución del programa entra en la función y el parámetro `name` se establece automáticamente en 'Alice', que se imprime con la declaración `print()` 2. Deberías usar parámetros en una función si necesitas que siga instrucciones ligeramente diferentes dependiendo de los valores que le pases en la llamada.

Hay algo digno de mención en los parámetros: el valor almacenado en un parámetro se olvida cuando vuelve la función. Por ejemplo, si añadiésemos `print(name)` después de `say_hello_to('Bob')` en el programa anterior, el programa daría error porque no hay una variable llamada `name`. Esta variable se destruye después de que vuelva la llamada a la función `say_hello_to('Bob')`, así que `print(name)` se referiría a una variable que no existe.

Los términos “definir”, “llamar”, “pasar”, “argumento” y “parámetro” pueden ser confusos. Para repasar su significado, vamos a ver un ejemplo de código:

```
1 def say_hello_to(name):
```

```
# Imprime tres saludos para el nombre proporcionado.
```

```
print('Good morning, ' + name)
```

```
print('Good afternoon, ' + name)
```

```
print('Good evening, ' + name)
```

```
2 say_hello_to('Al')
```

Definir una función es crearla, igual que una declaración de asignación como `spam = 42` crea la variable `spam`. La declaración `def` define la función `say_hello_to()` 1. La línea `say_hello_to('Al')` 2

llama a la función que hemos creado, enviando la ejecución al principio del código de la función. Esta llamada a la función le pasa la cadena 'Al'. Un valor pasado en una llamada a una función es un argumento. Los argumentos se asignan a variables locales denominadas parámetros. El argumento 'Al' se asigna al parámetro name. Es fácil mezclar estos términos, pero tenerlos claros te ayudará a saber exactamente lo que significa el texto en este capítulo.

Valores de retorno y declaraciones return

Cuando llamamos a la función len() y le pasamos un argumento como 'Hello', la llamada a la función se evalúa como el valor entero 5, que es la longitud de la cadena que le hemos pasado. En general, el valor al que se evalúa una llamada a una función se denomina valor de retorno de la función.

Cuando se crea una función con la declaración def, se puede especificar el valor de retorno con una declaración return, que consiste en lo siguiente:

- La palabra clave return.
- El valor o expresión que debería devolver la función.

En el caso de una expresión, el valor de retorno es a lo que se evalúe esta expresión. Por ejemplo, el siguiente programa define una función que devuelve una cadena diferente dependiendo del número que se le pase como argumento. Escribe este código en el editor de archivos y guárdalo como magic8Ball.py:

```
1 import random

2 def get_answer(answer_number):

    # Devuelve una predicción basándose en el entero que sea
    answer_number, del 1 al 9

3 if answer_number == 1:

    return 'It is certain'
```

```
elif answer_number == 2:
    return 'It is decidedly so'
elif answer_number == 3:
    return 'Yes'
elif answer_number == 4:
    return 'Reply hazy try again'
elif answer_number == 5:
    return 'Ask again later'
elif answer_number == 6:
    return 'Concentrate and ask again'
elif answer_number == 7:
    return 'My reply is no'
elif answer_number == 8:
    return 'Outlook not so good'
elif answer_number == 9:
    return 'Very doubtful'

print('Ask a yes or no question:')
input('>')

4 r = random.randint(1, 9)
5 fortune = get_answer(r)
6 print(fortune)
```

Cuando empieza el programa, Python importa, en primer lugar, el

módulo `random` 1. Después, viene la definición de la función `get_answer()` 2. Puesto que no se está llamando a la función, no se ejecuta el código de dentro. A continuación, el programa llama a la función `random.randint()` con dos argumentos: 1 y 9 4. Esta función evalúa un entero aleatorio entre 1 y 9 (ambos incluidos) y lo almacena en una variable llamada `r`. Ahora el programa llama a la función `get_answer()` con `r` como argumento 5. La ejecución del programa se mueve hasta el principio de esa función 3, almacenando el valor `r` en un parámetro llamado `answer_number`. Entonces, dependiendo del valor en `answer_number`, la función devuelve uno de varios valores de cadena posibles. La ejecución devuelve la línea al final del programa que llamó originalmente a `get_answer()` 5 y asigna la cadena devuelta a una variable llamada `fortune`, que después se pasa a una llamada a `print()` 6 y se muestra en la pantalla.

Observa que, como podemos pasar valores de retorno como argumentos a otras llamadas a funciones, podríamos reducir estas tres líneas:

```
r = random.randint(1, 9)
```

```
fortune = get_answer(r)
```

```
print(fortune)
```

a esta única línea equivalente:

```
print(get_answer(random.randint(1, 9)))
```

Recuerda que las expresiones consisten en valores y operadores; podemos usar una llamada a una función en una expresión porque la llamada se evalúa como su valor de retorno.

El valor `None`

En Python, un valor llamado `None` representa la ausencia de un valor. El valor `None` es el único valor del tipo de datos `NoneType`. (Otros lenguajes de programación podrían llamarlo `null`, `nil` o `undefined`). Igual que pasa con los valores booleanos `True` y `False`, siempre hay que escribir `None` con mayúscula inicial.

Este valor sin un valor puede ser útil cuando hay que guardar algo que no debería confundirse con un valor real en una variable. Un lugar donde se utiliza None es como valor de retorno de print(). La función print() muestra texto en la pantalla y no necesita devolver nada como sí requieren len() o input(). Pero, dado que todas las llamadas a funciones necesitan evaluarse como un valor de retorno, print() devuelve None. Para ver esto en acción, introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > spam = print('Hello!')
```

Hello!

```
> > > None = spam
```

True

Entre bambalinas, Python añade return None al final de cualquier definición de función sin declaración return. Este comportamiento se parece a la manera en la que un bucle while o for termina implícitamente con una declaración continue. Las funciones también devuelven None si se usa una declaración return sin un valor (es decir, la palabra clave return a palo seco).

Parámetros con nombre

Python identifica la mayoría de los argumentos por su posición en la llamada a la función. Por ejemplo, random.randint(1, 10) no es lo mismo que random.randint(10, 1). La primera llamada devuelve un entero aleatorio entre 1 y 10 porque el primer argumento determina el extremo inferior del rango y el siguiente el superior, mientras que la segunda llamada provoca un error.

Por otro lado, Python identifica los parámetros con nombre por el nombre colocado delante de ellos en la llamada a la función. A veces también se llama a estos parámetros con nombre parámetros o argumentos de palabra clave, aunque no tienen nada que ver con las palabras clave de Python. Los programadores emplean con frecuencia parámetros con nombre para dar argumentos opcionales. Por ejemplo, la función print() utiliza los parámetros opcionales end y sep para especificar caracteres separadores que imprimir al final

de sus argumentos y entre ellos, respectivamente. Si ejecutásemos un programa sin estos argumentos:

```
print('Hello')
```

```
print('World')
```

la salida sería así:

```
Hello
```

```
World
```

Las dos cadenas aparecen en líneas separadas porque la función `print()` añade automáticamente un carácter de salto de línea al final de la cadena que se le pasa.

Sin embargo, podemos hacer que el parámetro `end` cambie ese salto de línea por una cadena diferente. Por ejemplo, si el código fuese este:

```
print('Hello', end = '')
```

```
print('World')
```

la salida sería así:

```
HelloWorld
```

La salida aparece en una sola línea porque Python ya no imprime el salto después de 'Hello'. En su lugar, muestra una cadena en blanco. Esto es útil si hay que deshabilitar el salto de línea que se añade al final de cada llamada a la función `print()`. Supongamos que queremos imprimir los resultados de cara (*heads*, H) o cruz (*tails*, T) de una serie de lanzamientos de moneda. Imprimir la salida en una sola línea hace la salida más bonita, como en este programa `coinflip.py`:

```
import random
```

```
for i in range(100): # Realiza 100 lanzamientos de moneda.
```

```
if random.randint(0, 1) == 0:
```

```
    print('H', end='')
```

```
else:
```

```
    print('T', end='')
```

```
print() # Imprime una nueva línea al final.
```

Este programa muestra los resultados H y T en una línea compacta, en vez de expandirlos con un resultado H o T por línea:

```
T H T T T H H T T T T H H H H T H H T T T T T H T T T T T H T T
T T T H T H T

H H H T T H T T T T H T H H H T H H T H T T T T T H T T H T T T
T H T H H H T

T T T H T T T T H H H T H T H H H H T H H T
```

Del mismo modo, cuando pasamos varios valores de cadena a `print()`, la función los separa automáticamente con un único espacio. Para ver este comportamiento, escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > print('cats', 'dogs', 'mice')
```

```
cats dogs mice
```

Podrías sustituir la cadena de separación predeterminada pasando al parámetro `sep` una cadena diferente. Introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > print('cats', 'dogs', 'mice', sep=',')
```

```
cats,dogs,mice
```

También puedes añadir parámetros con nombre a las funciones que escribas tú, pero, antes, debes aprender sobre los tipos de datos lista y diccionario en los capítulos 6 y 7. Por ahora, basta con que sepas que algunas funciones tienen parámetros con nombre opcionales

que puedes especificar al llamar a la función.

La pila de llamadas

Imagina haber tenido una conversación dispersa con alguien. Estabais hablando de tu amiga Alice, lo cual te recordó una historia sobre tu compañero de trabajo Bob, pero primero tenías que explicar una cosa de tu prima Carol. Al acabar la historia sobre Carol, volviste a hablar de Bob y, cuando terminaste la historia de Bob, volviste a hablar de Alice. Pero entonces te acordaste de tu hermano David, así que contaste una historia sobre él y luego volviste a Alice para terminar esa historia. Tu conversación siguió una estructura de pila, como en la figura 4.1. En una pila, los elementos se añaden o eliminan de la parte superior solo y el tema actual está siempre encima de la pila.

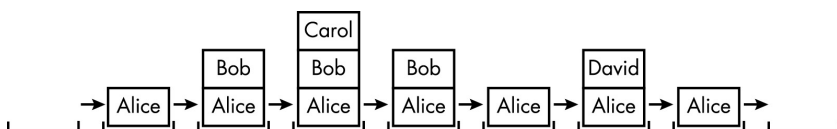


Figura 4.1. La pila de tu conversación dispersa.

Igual que con tu conversación dispersa, llamar a una función no envía la ejecución en un viaje solo de ida a la parte superior de una función. Python recordará qué línea de código llamó a la función para que la ejecución pueda volver ahí cuando encuentre una declaración `return`. Si esa función original hubiese llamado a otras funciones, la ejecución volvería a esas llamadas primero, antes de volver a la llamada original. La llamada a la función que está encima de la pila es la ubicación actual de la ejecución.

Abre una ventana del editor de archivos y escribe el siguiente código, guardándolo como `abcdCallStack.py`:

```
def a():  
    print('a() starts')  
1 b()  
2 d()
```

```
print('a() returns')
```

```
def b():
```

```
    print('b() starts')
```

```
    3 c()
```

```
    print('b() returns')
```

```
def c():
```

```
    4 print('c() starts')
```

```
    print('c() returns')
```

```
def d():
```

```
    print('d() starts')
```

```
    print('d() returns')
```

```
5 a()
```

Si ejecutas el programa, la salida tendrá este aspecto:

```
a() starts
```

```
b() starts
```

```
c() starts
```

```
c() returns
```

```
b() returns
```

```
d() starts
```

```
d() returns
```

```
a() returns
```

Cuando se llama a a() 5, esta llama a b() 1, que, a su vez, llama a

c() 3. La función c() no llama a nada, solo muestra c() starts 4 y c() returns antes de volver a la línea de b() que la ha llamado 3. Una vez que la ejecución vuelve al código de b() que ha llamado a c(), vuelve a la línea de a() que ha llamado a b() 1. La ejecución continúa hasta la siguiente línea de la función a() 2, que es una llamada a d(). Al igual que la función c(), d() tampoco llama a nada, solo muestra d() starts y d() returns antes de volver a la línea de a() que la ha llamado. Como d() no contiene más código, la ejecución vuelve a la línea de a() que la ha llamado 2. La última línea de a() muestra a() returns antes de volver a la llamada original a a() al final del programa 5.

La pila de llamadas es la manera en que Python recuerda dónde debe volver la ejecución después de cada llamada a una función. La pila de llamadas no se guarda en una variable del programa, sino que es una sección de la memoria del ordenador que Python maneja automáticamente entre bambalinas. Cuando el programa llama a una función, Python crea un objeto marco en la parte superior de la pila de llamadas. Los objetos marco almacenan el número de línea de la llamada a la función original de modo que Python pueda recordar dónde volver. Si el programa llama a otra función, Python añade otro objeto marco por encima del otro de la pila de llamadas.

Cuando vuelve una llamada a una función, Python quita el objeto marco de encima de la pila y mueve la ejecución al número de línea guardado en ella. Observa que los objetos marco siempre se añaden y eliminan en la parte superior de la pila y no en cualquier otro lugar. La figura 4.2 ilustra el estado de las llamadas en abcdCallStack.py cuando se llama a cada función y vuelve.

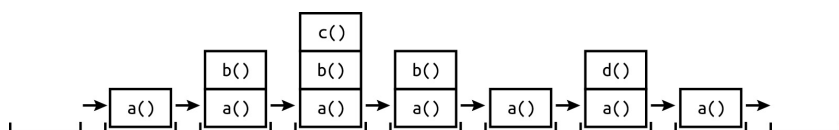


Figura 4.2. Los objetos marco de la pila de llamadas a medida que abcdCallStack.py llama a funciones y vuelve.

La parte superior de la pila de llamadas es la función que se está ejecutando en ese momento. Cuando se vacía la pila de llamadas, la ejecución está en una línea fuera de todas las funciones. La pila de

llamadas es un detalle técnico sobre el que no es estrictamente necesario saber para escribir programas. Basta con entender que las llamadas a funciones vuelven al número de línea desde el que se llamaron. Sin embargo, comprender las pilas de llamadas hace que sea más fácil entender los ámbitos locales y globales de los que hablaremos en el siguiente apartado.

Ámbito local y global

Solo el texto dentro de una función puede acceder a los parámetros y variables asignados en esa función. Se dice que estas variables existen en el ámbito local de esa función. Por el contrario, el código escrito en cualquier parte de un programa puede acceder a variables que estén asignadas fuera de las funciones. Estas variables existen en el ámbito global. Una variable que existe en un ámbito local se denomina variable local, mientras que una variable que existe en un ámbito global es una variable global. Una variable debe ser de un tipo o de otro, no puede ser a la vez local y global.

Imagina el ámbito como un contenedor para variables. Solo hay un ámbito global, creado cuando empieza el programa. Cuando el programa termina, destruye el ámbito global y todas sus variables se olvidan. Cuando el programa llama a una función, se crea un nuevo ámbito local. Cualquier variable asignada en la función, existe dentro del ámbito local de esa función. Cuando vuelve la función, el ámbito local se destruye junto con esas variables.

Python utiliza los ámbitos porque permiten a una función modificar sus variables y seguir interactuando con el resto del programa a través de sus parámetros y su valor de retorno únicamente. Esto reduce el número de líneas de código que podrían estar causando un error. Si un programa solo contiene variables globales y un error hace que una variable se establezca en un valor incorrecto, puede ser difícil localizar ese valor. Podría estar en cualquier parte del programa, que podría tener cientos o incluso miles de líneas. Pero, si el error se produce en una variable local, podemos restringir la búsqueda a una única función.

Por este motivo, aunque no está mal usar variables globales en programas pequeños, no es buena idea depender de variables

globales cuando tus programas se vayan haciendo más grandes.

Reglas de ámbito

Cuando trabajes con variables locales y globales, ten en cuenta las siguientes reglas:

- El código que está en el ámbito global, fuera de todas las funciones, no puede emplear variables locales.
- El código que está en el ámbito local de una función no puede utilizar variables de cualquier otro ámbito local.
- El código en un ámbito local puede acceder a variables globales.
- Se puede usar el mismo nombre para distintas variables si estas están en distintos ámbitos. Esto es, puede haber una variable local llamada spam y una variable global que también se llame spam.

Vamos a revisar estas reglas con ejemplos.

El código que está en el ámbito global no puede emplear variables locales

Fíjate en el siguiente código, que dará un error cuando lo ejecutes:

```
def spam():
```

```
    1 eggs = 'sss'
```

```
    spam()
```

```
    print(eggs)
```

La salida del programa tendrá este aspecto:

Traceback (most recent call last):

File "C:/test1.py", line 4, in <module>

```
    print(eggs)
```

NameError: name 'eggs' is not defined

El error se produce porque la variable `eggs` existe solo en el ámbito local creado cuando se llama a `spam()` 1. Una vez que la ejecución del programa vuelve de `spam()`, ese ámbito local se destruye y ya no hay ninguna variable llamada `eggs`. Así pues, cuando el programa intenta ejecutar `print(eggs)`, Python da un error indicando que `eggs` no está definida. Esto tiene sentido si lo piensas; cuando la ejecución del programa está en el ámbito global, no existe ningún ámbito local, así que no puede haber variables locales. Por eso, solo se puede hacer referencia a variables globales en el ámbito global.

El código que está en un ámbito local no puede utilizar variables de otro ámbito local

Python crea un nuevo ámbito local cada vez que un programa llama a una función, incluso cuando se llama a esa función desde otra función. Observa este programa:

```
def spam():  
  
    eggs = 'SPAMSPAM'  
  
    1 bacon()  
  
    2 print(eggs) # Imprime 'SPAMSPAM'  
  
def bacon():  
  
    ham = 'hamham'  
  
    eggs = 'BACONBACON'  
  
    3 spam()
```

Cuando empieza el programa, llama a la función `spam()` 3, creando un ámbito local. La función `spam()` establece la variable local `eggs` en `'SPAMSPAM'` y luego llama a la función `bacon()` 1, creando un segundo ámbito local. Pueden existir varios ámbitos locales al mismo tiempo. En este nuevo ámbito local, la variable local `ham` se establece en `'hamham'` y se crea una variable local `eggs` (distinta de la del ámbito local de `spam()`), que se establece en `'BACONBACON'`. En este punto, el programa tiene dos variables locales llamadas `eggs` que existen simultáneamente: una es local para `spam()` y la otra es

local para bacon().

Cuando vuelve bacon(), Python destruye el ámbito local para esa llamada, incluida su variable eggs. La ejecución del programa continúa en la función spam(), imprimiendo el valor de eggs 2. Como el ámbito local de la llamada a spam() todavía existe, la única variable eggs es la de la función spam(), que se estableció en 'SPAMSPAM'. Eso es lo que imprime el programa.

El código que está en un ámbito local puede usar variables globales

Hasta ahora, hemos visto que el código del ámbito local no puede acceder a variables de un ámbito local, igual que tampoco puede el código de otro ámbito local. Ahora observa el siguiente programa:

```
def spam():  
    print(eggs) # Imprime 'GLOBALGLOBAL'  
  
eggs = 'GLOBALGLOBAL'  
  
spam()  
  
print(eggs)
```

Como la función spam() no tiene un parámetro llamado eggs ni código que asigne un valor a eggs, Python considera el uso que hace la función de eggs una referencia a la variable global eggs. Por eso el programa imprime 'GLOBALGLOBAL' al ejecutarse.

Las variables locales y globales pueden tener el mismo nombre

Técnicamente, es aceptable usar el mismo nombre para una variable global y variables locales en distintos ámbitos. Pero, para no complicarte la vida, evítalo. Para ver lo que podría pasar, escribe el siguiente código en el editor de archivos y guárdalo como localGlobalSameName.py:

```
def spam():  
  
    1 eggs = 'spam local'
```

```
print(eggs) # Imprime 'spam local'

def bacon():
    2 eggs = 'bacon local'

    print(eggs) # Imprime 'bacon local'

    spam()

    print(eggs) # Imprime 'bacon local'

    3 eggs = 'global'

    bacon()

    print(eggs) # Imprime 'global'
```

Esta es la salida al ejecutar el programa:

bacon local

spam local

bacon local

global

Este programa contiene en realidad tres variables diferentes, pero todas se llaman eggs y eso resulta confuso. Estas son las variables:

- Una variable eggs que existe en un ámbito local cuando se llama a spam() 1.
- Una variable eggs que existe en un ámbito local cuando se llama a bacon() 2.
- Una variable eggs que existe en el ámbito global 3.

Como estas tres variables distintas tienen el mismo nombre, puede ser difícil saber cuál se está usando en un momento dado. Es mejor dar a las variables nombres únicos incluso aunque estén en ámbitos diferentes.

La declaración global

Si necesitas modificar una variable global desde dentro de una función, utiliza la declaración global. Incluir una línea como `global eggs` al principio de una función dice a Python: “En esta función, `eggs` se refiere a la variable global, así que no crees una variable local con este nombre”. Para ver un ejemplo, escribe el siguiente código en el editor de archivos y guárdalo como `globalStatement.py`:

```
def spam():  
  
    1 global eggs  
  
    2 eggs = 'spam'  
  
    eggs = 'global'  
  
    spam()  
  
    print(eggs) # Imprime 'spam'
```

Cuando ejecutes el programa, la última llamada a `print()` tendrá esta salida:

```
spam
```

Dado que se ha declarado `eggs` como global al principio de `spam()` 1, establecer `eggs` en `'spam'` 2 cambia el valor de `eggs` en el ámbito global. Nunca se crea una variable local `eggs`.

Identificación del ámbito

Usa estas cuatro reglas para determinar si una variable pertenece a un ámbito local o al ámbito global:

1. Una variable del ámbito global (es decir, fuera de cualquier función) es siempre una variable global.
2. Una variable en una función con una declaración global es siempre una variable global en esa función.

3. De lo contrario, si una función usa una variable en una declaración de asignación, es una variable local.

4. No obstante, si la función utiliza una variable pero no en una declaración de asignación, es una variable global.

Aquí tienes un programa de ejemplo para hacerte mejor idea de estas reglas. Introduce el siguiente código en el editor de archivos y guárdalo como `sameNameLocalGlobal.py`:

```
def spam():  
    1 global eggs  
  
    eggs = 'spam' # Esta es la variable global.  
  
def bacon():  
    2 eggs = 'bacon' # Esta es una variable local.  
  
def ham():  
    3 print(eggs) # Esta es la variable global.  
  
eggs = 'global' # Esta es la variable global.  
  
spam()  
  
print(eggs)
```

En la función `spam()`, `eggs` se refiere a la variable global `eggs` porque la función incluye una declaración global para ella 1. En `bacon()`, `eggs` es una variable local porque la función incluye una declaración de asignación para ella 2. En `ham()` 3, `eggs` es la variable global porque la función no contiene declaración de asignación ni global para la variable. Si ejecutas `sameNameLocalGlobal.py`, la salida será así:

```
spam
```

Si intentas usar una variable local en una función antes de asignarle un valor, como en el siguiente programa, Python dará error. Para

verlo, escribe lo siguiente en el editor de archivos y guárdalo como `sameNameError.py`:

```
def spam():  
  
    print(eggs) # ¡ERROR!  
  
    1 eggs = 'spam local'  
  
    2 eggs = 'global'  
  
spam()
```

Si ejecutas el programa anterior, produce un mensaje de error:

Traceback (most recent call last):

File "C:/sameNameError.py", line 6, in <module>

spam()

File "C:/sameNameError.py", line 2, in spam

print(eggs) # ¡ERROR!

UnboundLocalError: local variable 'eggs' referenced before assignment

Este error se produce porque Python ve que hay una declaración de asignación para `eggs` en la función `spam()` 1 y, por lo tanto, considera que cualquier mención a una variable `eggs` en `spam()` es una variable local. Pero, como `print(eggs)` se ejecuta antes de que se asigne nada a `eggs`, la variable local `eggs` no existe. Python no retrocederá para usar la variable global `eggs` 2.

El error `UnboundLocalError` indica que el programa ha utilizado una variable local antes de que se le asignase un valor.

Las funciones como “cajas negras”

A menudo, lo único que necesitamos saber de una función es su entrada (los parámetros) y su valor de salida; no hay que cargar siempre con el conocimiento de cómo funciona el código de la

función. Cuando pensamos en las funciones de esta forma, es habitual decir que tratamos una función como una “caja negra”.

Esta idea es fundamental para la programación moderna. En capítulos posteriores te mostraré varios módulos con funciones escritas por otras personas. Aunque puedes echar un vistazo al código fuente, si tienes curiosidad, no te hace falta saber cómo funcionan para poder usarlas. Y, como es preferible escribir funciones sin variables globales, por lo general no tendrás que preocuparte por que el código de la función interactúe con el resto del programa.

Manejo de excepciones

Ahora mismo, un error o excepción en tu programa de Python implica que falle todo el programa. No queremos que esto ocurra en los programas de verdad. Lo que nos interesa es que el programa detecte errores, los maneje y siga ejecutándose.

Por ejemplo, fíjate en el siguiente programa, que tiene un error de división entre cero. Abre una ventana del editor de archivos e introduce el siguiente código, guardándolo como `zeroDivide.py`:

```
def spam(divide_by):  
  
    return 42 / divide_by  
  
print(spam(2))  
  
print(spam(12))  
  
print(spam(0))  
  
print(spam(1))
```

Hemos definido una función llamada `spam`, le hemos dado un parámetro y se ha imprimido el valor de esa función con varios parámetros para ver qué pasa. Esta es la salida que recibirás al ejecutar el código anterior:

3.5

Traceback (most recent call last):

File "C:/zeroDivide.py", line 6, in <module>

```
print(spam(0))
```

File "C:/zeroDivide.py", line 2, in spam

```
return 42 / divide_by
```

ZeroDivisionError: division by zero

El error `ZeroDivisionError` se produce siempre que se intenta dividir un número entre cero. Por el número de línea que indica el mensaje de error, sabemos que la declaración `return` de `spam()` es la que produce el error.

La mayor parte del tiempo, las excepciones indican un error en el código que hay que corregir. Pero a veces pueden ser esperadas y se pueden resolver. Por ejemplo, en el capítulo 10 veremos cómo leer texto de archivos. Si especificamos un nombre para un archivo que no existe, Python creará una excepción `FileNotFoundError`. Es posible que quieras manejar esta excepción pidiendo al usuario que vuelva a introducir el nombre de archivo en lugar de dejar que esta excepción sin manejar haga fallar de inmediato el programa.

Los errores se pueden manejar con declaraciones `try` y `except`. El código que podría contener un error se pone en una proposición `try`. La ejecución del programa se mueve al inicio de la siguiente proposición `except` si se produce un error.

Podemos poner el código de división entre cero anterior en una proposición `try` y tener una proposición `except` con código para manejar lo que sucede cuando se produce este error:

```
def spam(divide_by):
```

```
    try:
```

```
        # Cualquier código en este bloque que provoque ZeroDivisionError
```


no hará fallar el programa:

```
return 42 / divide_by
```

```
except ZeroDivisionError:
```

Si se produce ZeroDivisionError, se ejecuta el código de este bloque:

```
print('Error: Invalid argument.')
```

```
print(spam(2))
```

```
print(spam(12))
```

```
print(spam(0))
```

```
print(spam(1))
```

Cuando el código de una proposición try provoca un error, la ejecución del programa pasa automáticamente al código de la proposición except. Después de ese código, la ejecución continúa con normalidad. Si el programa no lanza una excepción en la proposición try, se salta el código de la proposición except. La salida del programa anterior sería así:

21.0

3.5

Error: Invalid argument.

None

42.0

Cualquier error que se produzca en las llamadas a funciones del bloque try será interceptado. Considera el siguiente programa, que tiene llamadas a spam() en el bloque try:

```
def spam(divide_by):
```

```
return 42 / divide_by
```

try:

```
print(spam(2))
```

```
print(spam(12))
```

```
print(spam(0))
```

```
print(spam(1))
```

except ZeroDivisionError:

```
print('Error: Invalid argument.')
```

Al ejecutar este programa, la salida tendrá este aspecto:

21.0

3.5

Error: Invalid argument.

La razón por la que `print(spam(1))` no se ejecuta es que, una vez que la ejecución salta al código de la proposición `except`, ya no vuelve a la proposición `try`, sino que sigue avanzando por el programa con normalidad.

Un programa corto: zigzag

Vamos a usar los conceptos de programación aprendidos hasta ahora para crear un pequeño programa de animación. Este programa creará un patrón en zigzag hasta que el usuario lo detenga haciendo clic en el botón Detener del editor Mu o pulsando **Control-C**. Cuando ejecutes el programa, la salida será algo así:

```
*****
```

```
*****
```

```
*****
```

```
*****
```

Escribe el siguiente código fuente en el editor de archivos y guarda como zigzag.py:

```
import time, sys

indent = 0 # Espacios para sangrado

indent_increasing = True # Si el sangrado aumenta o no

try:

while True: # Bucle principal del programa

print(' ' * indent, end='')

print('*****')

time.sleep(0.1) # Pausa de una décima de segundo

if indent_increasing:

# Aumenta el número de espacios:

indent = indent + 1

if indent == 20:

# Cambia la dirección:

indent_increasing = False

else:
```

Disminuye el número de espacios:

```
indent = indent - 1
```

```
if indent == 0:
```

Cambia la dirección:

```
indent_increasing = True
```

```
except KeyboardInterrupt:
```

```
sys.exit()
```

Echemos un vistazo a este código línea por línea, desde el principio:

```
import time, sys
```

```
indent = 0 # Espacios para sangrado
```

```
indent_increasing = True # Si el sangrado aumenta o no
```

Primero, importamos los módulos `time` y `sys`. Nuestro programa usa dos variables: `indent` lleva la cuenta de espacios de sangrado antes de la tanda de ocho asteriscos e `indent_increasing` contiene un valor booleano que determina si el sangrado aumenta o disminuye:

```
try:
```

```
while True: # Bucle principal del programa
```

```
print(' ' * indent, end='')
```

```
print('*****')
```

```
time.sleep(0.1) # Pausa de una décima de segundo
```

A continuación, ponemos el resto del programa dentro de una declaración `try`. Cuando el usuario pulsa **Control-C** mientras se está ejecutando un programa, Python lanza la excepción `KeyboardInterrupt`. Si no hay declaración `try-except` para interceptar esta excepción, el programa falla con un mensaje de error. Sin embargo, queremos que nuestro programa maneje

limpiamente la excepción KeyboardInterrupt llamando a sys.exit(). (El código que hace esto está en la declaración except, al final del programa).

El bucle infinito while True: repetirá las instrucciones del programa para siempre. Esto implica usar ‘ ‘ * indent para imprimir el número correcto de espacios para el sangrado. No queremos imprimir automáticamente un salto de línea después de estos espacios, así que también pasamos end=” to a la primera llamada a print(). Una segunda llamada a print() imprime la línea de asteriscos. Todavía no hemos comentado la función time.sleep(); basta con saber que introduce una pausa de una décima de segundo en nuestro programa:

```
if indent_increasing:
```

```
# Aumenta el número de espacios:
```

```
indent = indent + 1
```

```
if indent == 20:
```

```
indent_increasing = False # Cambia la dirección
```

A continuación, ajustamos la cantidad de sangrado que se usará en la siguiente impresión de asteriscos. Si indent_increasing es True, añadiremos 1 a indent, pero, cuando indent llegue a 20, disminuirá el sangrado:

```
else:
```

```
# Disminuye el número de espacios:
```

```
indent = indent - 1
```

```
if indent == 0:
```

```
indent_increasing = True # Cambia la dirección
```

Si indent_increasing es False, restaremos uno a indent. Cuando indent llegue a 0, el sangrado volverá a aumentar. En cualquier caso, la ejecución volverá al inicio del bucle principal del programa

para imprimir más asteriscos.

Si el usuario pulsa **Control-C** en cualquier momento en el que la ejecución del programa esté en el bloque try, se lanza esta declaración except, que maneja la excepción KeyboardInterrupt:

except KeyboardInterrupt:

sys.exit()

La ejecución del programa entra en el bloque except, que ejecuta sys.exit() y sale del programa. De este modo, aunque el bucle sea infinito, el usuario puede cerrar el programa.

Un programa corto: espiga

Vamos a crear otro programa animado. Este utiliza replicación de cadenas y bucles anidados para dibujar espigas. Abre un archivo en tu editor de código, escribe lo siguiente y guarda como spike.py:

```
import time, sys
```

```
try:
```

```
while True: # Bucle principal del programa
```

```
# Dibuja líneas con una longitud cada vez mayor:
```

```
for i in range(1, 9):
```

```
print('-' * (i * i))
```

```
time.sleep(0.1)
```

```
# Dibuja líneas con una longitud cada vez menor:
```

```
for i in range(7, 1, -1):
```

```
print('-' * (i * i))
```

```
time.sleep(0.1)
```


Dibuja líneas con una longitud cada vez mayor:

```
for i in range(1, 9):
```

```
    print('-' * (i * i))
```

```
    time.sleep(0.1)
```

Como la variable *i* está establecida en 1, luego en 2, después en 3, y así sucesivamente hasta 9 (sin incluirlo), la siguiente llamada a `print()` replica las cadenas '-' por $1 * 1$ (esto es, 1), luego $2 * 2$ (o sea, 4), luego $3 * 3$ (es decir, 9), etc. Este código es lo que crea las cadenas que tienen 1, 4, 9, 16, 25, 36, 49 y 64 guiones de largo. Al tener cadenas exponencialmente más largas, creamos la parte superior de la espiga.

Para dibujar la mitad inferior, necesitamos otro bucle `for` que haga que *i* empiece en 7 y disminuya hasta 1, sin incluirlo:

Dibuja líneas con una longitud cada vez menor:

```
for i in range(7, 1, -1):
```

```
    print('-' * (i * i))
```

```
    time.sleep(0.1)
```

Puedes modificar los valores 9 y 7 en los dos bucles `for` si quieres cambiar la anchura de amplitud de la línea. El resto del código seguiría funcionando igual de bien con los nuevos valores.

Resumen

Las funciones son la principal forma de compartimentar el código en grupos lógicos. Dado que las variables de las funciones están en su propio ámbito local, el código de una función no puede afectar directamente a los valores de las variables locales de otras funciones. Esto limita las secciones de código que pueden modificar los valores de las variables, lo cual puede ser útil a la hora de depurar.

Las funciones son una herramienta estupenda para ayudarte a

organizar el código. Imagínalas como cajas negras: tienen entradas en forma de parámetros y salidas en forma de valores de retorno y el código de dentro no afecta a las variables de otras funciones.

En capítulos anteriores, un solo error podía hacer que el programa fallase. En este capítulo, has conocido las declaraciones `try` y `except`, que ejecutan código cuando se detecta un error. Esto puede hacer tus programas más resilientes en casos habituales de error.

Preguntas prácticas

1. ¿Por qué son beneficiosas las funciones para tus programas?
2. ¿Cuándo se ejecuta el código de una función: cuando se define la función o cuando se la llama?
3. ¿Qué declaración crea una función?
4. ¿Cuál es la diferencia entre una función y una llamada a una función?
5. ¿Cuántos ámbitos globales hay en un programa de Python? ¿Y cuántos locales?
6. ¿Qué pasa con las variables de un ámbito local cuando vuelve la llamada a la función?
7. ¿Qué es un valor de retorno? ¿Puede un valor de retorno formar parte de una expresión?
8. Si una función no tiene declaración `return`, ¿cuál es el valor de retorno de una llamada a esa función?
9. ¿Cómo se puede obligar a una variable de una función a hacer referencia a la variable global?
10. ¿Qué es el tipo de datos `None`?
11. ¿Que hace la declaración `import a_really_ourpetsnamederic`?
12. Si tuvieses una función llamada `bacon()` en un módulo llamado `spam`, ¿cómo la llamarías después de importar `spam`?

13.¿Cómo podemos evitar que un programa falle cuando se produzca un error?

14.¿Qué va dentro de la proposición try? ¿Y de except?

15.Escribe el siguiente programa en un archivo llamado notrandomdice.py y ejecútalo. ¿Por qué todas las llamadas a la función devuelven el mismo número?

```
import random

random_number = random.randint(1, 6)

def get_random_dice_roll():

    # Devuelve un entero aleatorio entre 1 y 6.

    return random_number

print(get_random_dice_roll())

print(get_random_dice_roll())

print(get_random_dice_roll())

print(get_random_dice_roll())
```

Programas prácticos

Para practicar, escribe programas que hagan las siguientes tareas.

La secuencia de Collatz

Escribe una función llamada collatz() que tenga un parámetro llamado number. Si number es par, collatz() debería imprimir number // 2 y devolver este valor. Si number es impar, collatz() debería imprimir y devolver return 3 * number + 1.

Después, escribe un programa que permita al usuario introducir un número y que llame a collatz() con ese número hasta que la función devuelva el valor 1. (Por increíble que parezca, esta secuencia funciona con cualquier entero; tarde o temprano, usando esta

secuencia llegará a 1. Ni siquiera los matemáticos están seguros de por qué. Tu programa está explorando la denominada conjetura de Collatz, también conocida como “el programa matemático más sencillo imposible”).

Recuerda convertir el valor de retorno de `input()` en un entero con la función `int()`; de lo contrario, será un valor de cadena. Para que la salida sea más compacta, las llamadas a `print()` que imprimen los números deberían tener un parámetro con nombre `end = ' '` para mostrar todos los valores en una sola línea.

La salida de este programa debería ser algo así:

Enter number:

3

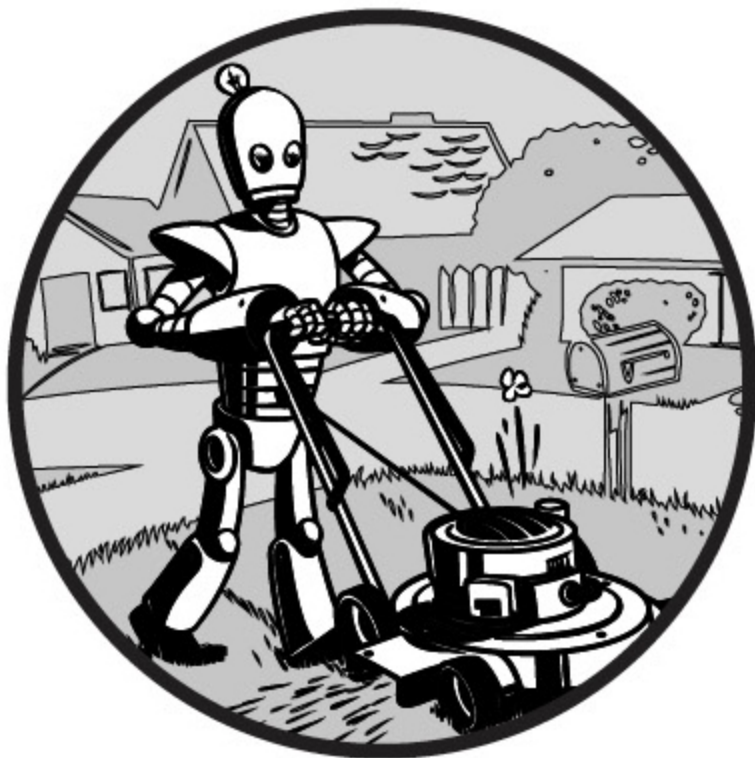
3 10 5 16 8 4 2 1

Pista: un número entero es par si `number % 2 == 0` e impar si `number % 2 == 1`.

Validación de la entrada

Añade declaraciones `try` y `except` al proyecto anterior para detectar si el usuario ha introducido una cadena no entera. Por lo general, la función `int()` dará un error `ValueError` si lo que se le pasa no es un entero, como en `int('puppy')`. En la proposición `except`, muestra un mensaje al usuario indicando que debe introducir un entero.

Depuración



Ahora que ya sabes lo suficiente para escribir programas básicos, es posible que empieces a encontrar fallos no tan básicos en ellos. Este capítulo presenta algunas herramientas y técnicas para llegar a la raíz de los errores de un programa y poder solucionarlos más deprisa y con menos esfuerzo. Parafraseando un viejo chiste de programadores, escribir código es un 90 por ciento de la programación; depurar código es el otro 90 por ciento.

El ordenador solo hace lo que le decimos que haga, no nos lee la mente para saber lo que queríamos que hiciera. Incluso los programadores profesionales crean fallos constantemente, así que

no te desanimes si tu programa tiene un problema.

Por suerte, hay unas cuantas herramientas y técnicas que pueden identificar exactamente qué está haciendo el código y dónde ha descarrilado. Vamos a utilizar el depurador, una función de Mu que ejecuta un programa instrucción a instrucción, dándonos la oportunidad de inspeccionar los valores de las variables mientras se ejecuta el código y ver cómo cambian a lo largo del programa. Este proceso es mucho más lento que ejecutar el programa a toda velocidad, pero nos permite ver los valores reales del programa mientras se ejecuta, en lugar de tener que deducir cuáles son los valores a partir del código fuente.

También haremos que el programa lance excepciones personalizadas para indicar errores y aprenderás sobre registros y aserciones, dos herramientas que pueden ayudarte a detectar fallos pronto. En general, cuanto antes interceptes los fallos, más fácil será resolverlos.

Lanzar excepciones

Python lanza una excepción cuando intenta ejecutar código no válido. En el capítulo 4, hemos manejado excepciones de Python con las declaraciones `try` y `except` para que el programa pudiese recuperarse de excepciones anticipadas. Pero también podemos lanzar nuestras propias excepciones en nuestro código. Lanzar una excepción es una forma de decir “deja de ejecutar este código y mueve la ejecución del programa a la declaración `except`”.

Las excepciones se lanzan con una declaración `raise`, que consta de lo siguiente:

- La palabra clave `raise`.
- Una llamada a la función `Exception()`.
- Una cadena con mensaje de error útil que pasa a la función `Exception()`.

Por ejemplo, escribe lo siguiente en el intérprete de comandos interactivo:

> > > **raise Exception('This is the error message.')**

Traceback (most recent call last):

File "<pyshell#191>", line 1, in <module>

raise Exception('This is the error message.')

Exception: This is the error message.

Si no hay declaraciones try y except que cubran la declaración raise que ha lanzado la excepción, el programa falla y muestra el mensaje de error de la excepción.

A menudo, es el código que llama a la función que contiene la declaración raise, más que la función en sí, el que sabe cómo manejar una excepción. Eso significa que normalmente veremos una declaración raise dentro de una función y las declaraciones try y except en el código que llama a la función. Por ejemplo, abre una pestaña nueva en el editor de archivos, escribe el siguiente código y guarda el programa como boxPrint.py:

```
def box_print(symbol, width, height):  
    if len(symbol) != 1:  
        1 raise Exception("Symbol must be a single character string.")  
    if width <= 2:  
        2 raise Exception("Width must be greater than 2.")  
    if height <= 2:  
        3 raise Exception("Height must be greater than 2.")  
    print(symbol * width)  
    for i in range(height - 2):  
        print(symbol + (' ' * (width - 2)) + symbol)  
    print(symbol * width)
```

try:

```
box_print('*', 4, 4)
```

```
box_print('O', 20, 5)
```

```
box_print('x', 1, 3)
```

```
box_print('ZZ', 3, 3)
```

4 except Exception as err:

```
5 print('An exception happened: ' + str(err))
```

try:

```
box_print('ZZ', 3, 3)
```

except Exception as err:

```
print('An exception happened: ' + str(err))
```

Aquí, hemos definido una función `box_print()` que toma un carácter, una anchura y una altura y utiliza ese carácter para hacer un dibujito de una caja con esa anchura y esa altura. Esta forma de caja se imprime en la pantalla.

Supongamos que queremos que la función acepte solo un carácter y esperamos que la anchura y la altura sean mayores que 2. Añadimos declaraciones `if` para lanzar excepciones si no se cumplen estos requisitos. Después, al llamar a `box_print()` con varios argumentos, nuestra try-except manejará los argumentos inválidos.

Este programa utiliza la forma `except Exception as err` de la declaración `except 4`. Si `box_print()` devuelve un objeto `Exception` 1 2 3, esta declaración `except` lo almacenará en una variable llamada `err`. Después, podemos convertir el objeto `Exception` en una cadena pasándoselo a `str()` para producir un mensaje de error más comprensible para el usuario 5. Al ejecutar `boxPrint.py`, la salida debería ser así:

* *

* *

OOOOOOOOOOOOOOOOOOOOOO

O O

O O

O O

OOOOOOOOOOOOOOOOOOOOOO

An exception happened: Width must be greater than 2.

An exception happened: Symbol must be a single character string.

Con las declaraciones try y except, podemos manejar los errores bien, en vez de dejar que falle todo el programa.

Aserciones

Una aserción es como una prueba de cordura para asegurarse de que el código no está haciendo nada que esté descaradamente mal. Realizamos estas comprobaciones con declaraciones assert. Si la prueba falla, el código lanza una excepción AssertionError. Una declaración assert consta de lo siguiente:

- La palabra clave assert.
- Una condición (es decir, una expresión que se evalúa como True o False).
- Una coma.
- Una cadena que se muestre cuando la condición sea False.

En lenguaje natural, una declaración assert dice “afirmo que la condición es verdadera y, si no lo es, hay algún fallo en alguna

parte, así que detén el programa de inmediato”. Para ver un ejemplo, introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > ages = [26, 57, 92, 54, 22, 15, 17, 80, 47, 73]
```

```
> > > ages.sort()
```

```
> > > ages
```

```
[15, 17, 22, 26, 47, 54, 57, 73, 80, 92]
```

```
> > > assert ages[0] <= ages[-1] # Afirma que la primera  
edad es <= que la última.
```

La declaración `assert` afirma que el primer elemento de `ages` debería ser menor o igual que el último. Esto es una prueba de cordura; si el código de `sort()` no tiene fallos y hace su trabajo, la aserción debería ser verdadera. Como la expresión `ages[0] <= ages[-1]` se evalúa como `True`, la declaración `assert` no hace nada.

Sin embargo, vamos a hacer como que hay un error en el código. Supongamos que por accidente llamamos al método de lista `reverse()` en lugar de `sort()`. Al escribir lo siguiente en el intérprete de comandos interactivo, la declaración `assert` da un error `AssertionError`:

```
> > > ages = [26, 57, 92, 54, 22, 15, 17, 80, 47, 73]
```

```
> > > ages.reverse()
```

```
> > > ages
```

```
[73, 47, 80, 17, 15, 22, 54, 92, 57, 26]
```

```
> > > assert ages[0] <= ages[-1] # Afirma que la primera  
edad es <= que la última.
```

Traceback (most recent call last):

File “<python-input-0>”, line 1, in <module>

AssertionError

A diferencia de las excepciones, el código no debería manejar las declaraciones `assert` con `try` y `except`; si una `assert` falla, el programa debería fallar. Con un “fallo rápido” como este, acortamos el tiempo entre la causa original del error y el momento en que lo notamos, reduciendo así la cantidad de código que hay que comprobar para dar con el origen del fallo.

Las aserciones son para errores del programador, no del usuario. Solo deberían fallar mientras se está desarrollando el programa; un usuario jamás debería encontrar un error de aserción en un programa terminado. Para los errores que el programa puede encontrar como parte normal de su funcionamiento (como que no se encuentra un archivo o el usuario haya introducido datos no válidos), lanza una excepción en lugar de detectarla con una declaración `assert`.

Registros

Si alguna vez has puesto una función `print()` en tu código para mostrar el valor de una variable mientras se ejecuta el programa, has usado una forma de registro para depurar tu código. Los registros son una buena manera de entender lo que está pasando en tu programa y en qué orden está sucediendo. El módulo `logging` de Python facilita la creación de un registro de mensajes personalizados que escribes tú. Estos mensajes de registro describirán cuándo la ejecución del programa ha llegado a la llamada a la función de registro y recogerá las variables especificadas en ese momento, ofreciendo un rastro que puede ayudarte a averiguar en qué momento ha empezado a ir mal la cosa. Por otro lado, un mensaje de registro ausente indica que se ha omitido una parte del código que nunca se ha ejecutado.

El módulo `logging`

Para permitir que el módulo `logging` muestre mensajes de registro en la pantalla cuando se ejecute el programa, copia lo siguiente encima del programa:

```
import logging
```

```
logging.basicConfig(level=logging.DEBUG, format='%(asctime)s -  
%(levelname)s - %(message)s')
```

La función `basicConfig()` del módulo `logging` permite especificar los detalles que queremos ver y cómo queremos que se muestren.

Supongamos que hemos escrito una función para calcular el factorial de un número. En matemáticas, el factorial de 4 es $1 \times 2 \times 3 \times 4$, o 24. El factorial de 7 es $1 \times 2 \times 3 \times 4 \times 5 \times 6 \times 7$, o 5.040. Abre una pestaña nueva en el editor de archivos y escribe el siguiente código. Tiene un error, pero vamos a generar varios mensajes de registro que nos ayudarán a saber qué va mal. Guarda el programa como `factorialLog.py`:

```
import logging
```

```
logging.basicConfig(level=logging.DEBUG, format='%(asctime)s -  
%(levelname)s - %(message)s')
```

```
logging.debug('Start of program')
```

```
def factorial(n):
```

```
    logging.debug('Start of factorial(' + str(n) + ')')
```

```
    total = 1
```

```
    for i in range(n + 1):
```

```
        total *= i
```

```
    logging.debug('i is ' + str(i) + ', total is ' + str(total))
```

```
    logging.debug('End of factorial(' + str(n) + ')')
```

```
    return total
```

```
print(factorial(5))
```

```
logging.debug('End of program')
```

Utilizamos la función `logging.debug()` para mostrar información del registro. Esta función `debug()` llama a `basicConfig()`, que imprime una línea de información en el formato especificado en la llamada a la función, junto con los mensajes que hemos pasado a `debug()`. La llamada `print(factorial(5))` forma parte del programa original, así que el código muestra el resultado incluso aunque los mensajes de registro estén deshabilitados. La salida de este programa tiene este aspecto:

```
2035-05-23 16:20:12,664 - DEBUG - Start of program
```

```
2035-05-23 16:20:12,664 - DEBUG - Start of factorial(5)
```

```
2035-05-23 16:20:12,665 - DEBUG - i is 0, total is 0
```

```
2035-05-23 16:20:12,668 - DEBUG - i is 1, total is 0
```

```
2035-05-23 16:20:12,670 - DEBUG - i is 2, total is 0
```

```
2035-05-23 16:20:12,673 - DEBUG - i is 3, total is 0
```

```
2035-05-23 16:20:12,675 - DEBUG - i is 4, total is 0
```

```
2035-05-23 16:20:12,678 - DEBUG - i is 5, total is 0
```

```
2035-05-23 16:20:12,680 - DEBUG - End of factorial(5)
```

```
0
```

```
2035-05-23 16:20:12,684 - DEBUG - End of program
```

La función `factorial()` devuelve 0 como factorial de 5, lo cual es incorrecto. El bucle `for` debería multiplicar el valor de `total` por los números de 1 a 5, pero el mensaje de error que muestra `logging.debug()` señala que la variable `i` empieza en 0 en vez de en 1. Dado que cero veces cualquier cosa es cero, el resto de las iteraciones tienen mal el valor para `total`. Cambia la línea `for i in range(n + 1):` por `for i in range(1, n + 1):` y vuelve a ejecutar el programa. La salida será así:

```
2035-05-23 17:13:40,650 - DEBUG - Start of program
```

2035-05-23 17:13:40,651 - DEBUG - Start of factorial(5)

2035-05-23 17:13:40,651 - DEBUG - i is 1, total is 1

2035-05-23 17:13:40,654 - DEBUG - i is 2, total is 2

2035-05-23 17:13:40,656 - DEBUG - i is 3, total is 6

2035-05-23 17:13:40,659 - DEBUG - i is 4, total is 24

2035-05-23 17:13:40,661 - DEBUG - i is 5, total is 120

2035-05-23 17:13:40,661 - DEBUG - End of factorial(5)

120

2035-05-23 17:13:40,666 - DEBUG - End of program

La llamada factorial(5) devuelve 120. Los mensajes del registro muestran lo que ha pasado dentro del bucle, lo que conduce directamente al error.

Vemos que las llamadas a logging.debug() no solo han imprimido las cadenas que se les han pasado, sino también un sello de tiempo y la palabra DEBUG.

Archivos de registro

En lugar de mostrar los mensajes de error en la pantalla, podemos escribirlos en un archivo de texto. La función logging.basicConfig() toma un parámetro llamado filename:

```
import logging
```

```
logging.basicConfig(filename = 'myProgramLog.txt',  
level = logging.DEBUG,
```

```
format = ' %(asctime)s - %(levelname)s - %(message)s')
```

Este código guardará los mensajes de error en myProgramLog.txt.

Aunque los mensajes de registro son útiles, pueden abarrotar la

pantalla y dificultar la lectura de la salida del programa. Escribir estos mensajes en un archivo mantendrá la pantalla limpia y te permitirá leer los mensajes después de ejecutar el programa. Puedes abrir este archivo en cualquier editor de texto, como el Bloc de notas o TextEdit.

Una mala práctica: depurar con print()

La introducción de `import logging` y `logging.basicConfig(level=logging.DEBUG, format='%%(asctime)s - %(levelname)s - %(message)s')` es un poco difícil de manejar. Puede que prefieras usar llamadas a `print()` en su lugar, ¡pero no caigas en esa trampa! Cuando hayas terminado de depurar, tendrás que dedicar un montón de tiempo a quitar llamadas a `print()` de tu código para cada mensaje de registro. Podrías incluso borrar llamadas a `print()` que no tengan nada que ver con esos mensajes. Lo bueno de los mensajes de registro es que eres libre de llenar tu programa con todos los que quieras y puedes deshabilitarlos luego añadiendo una simple llamada `logging.disable(logging.CRITICAL)`. A diferencia de `print()`, el módulo `logging` hace que sea fácil alternar entre mostrar y ocultar los mensajes de registro.

Los mensajes de registro están pensados para el programador, no para el usuario. El usuario no se preocupará por el contenido de un valor de diccionario que necesitas ver para ayudar con la depuración; usa un mensaje de registro para cosas como esa. Para los mensajes de error que debería ver el usuario, como “Archivo no encontrado” o “Entrada incorrecta, por favor introduzca un número”, utiliza una llamada a `print()`. No queremos privar al usuario de información útil que puede usar para resolver su problema.

Niveles de registro

Los niveles de registro ofrecen una manera de categorizar los mensajes de registro por importancia, de modo que podemos filtrar los que sean menos importantes mientras probamos los programas. Hay cinco niveles de registro, descritos en la tabla 5.1 de menos a más importante. El programa puede registrar mensajes en cada nivel con distintas funciones de registro.

Tabla 5.1. Niveles de registro en Python.

Función de registro

logging.debug() se usa para pequeños detalles. Por lo general, nos preocuparemos por estos mensajes solo cuando diagnostiquemos problemas.

logging.info() Registrar información sobre eventos generales del programa o confirmar que funciona en varios puntos.

logging.warning() Indicar un problema potencial que no impide que el programa funcione, pero podría acabar haciéndolo en el futuro.

logging.error() Registrar un error que ha hecho que el programa falle al hacer algo.

logging.critical(), utilizado para indicar un error fatal que ha hecho, o está a punto de hacer, que el programa deje de ejecutarse por completo.

Al final, depende de ti decidir en qué categoría entra tu mensaje de registro. Puedes pasar los mensajes de registro a estas funciones como una cadena. Prueba introduciendo lo siguiente en el intérprete de comandos interactivo:

```
> > > import logging
```

```
> > > logging.basicConfig(level=logging.DEBUG, format='%(asctime)s -
```

```
%(levelname)s - %(message)s')
```

```
> > > logging.debug('Some minor code and debugging details.')
```

```
2035-05-18 19:04:26,901 - DEBUG - Some debugging details.
```

```
> > > logging.info('An event happened.')
```

```
2035-05-18 19:04:35,569 - INFO - The logging module is working.
```

```
> > > logging.warning('Something could go wrong.')
```

```
2035-05-18 19:04:56,843 - WARNING - An error message is about to be logged.
```

```
> > > logging.error('An error has occurred.')
```

```
2035-05-18 19:05:07,737 - ERROR - An error has occurred.
```

```
> > > logging.critical('The program is unable to recover!')
```

```
2035-05-18 19:05:45,794 - CRITICAL - The program is unable to recover!
```

La ventaja de los niveles de registro es que se puede cambiar la prioridad de los mensajes que queremos ver. Pasar `logging.DEBUG` al parámetro `level` de la función `basicConfig()` mostrará mensajes de todos los niveles de registro (siendo `DEBUG` el más bajo). Pero, una vez que el programa está más o menos desarrollado, puede que solo nos interese ver los errores. En ese caso, podemos establecer el argumento `level` de `basicConfig()` en `logging.ERROR`. Así se mostrarán solo los mensajes `ERROR` y `CRITICAL` y se omitirán `DEBUG`, `INFO` y `WARNING`.

Registro deshabilitado

Una vez depurado el programa, no conviene que todos estos mensajes de registro saturen la pantalla. La función `logging.disable()` los deshabilita para no tener que quitar las llamadas de registro a mano. Basta con pasar un nivel de registro a `logging.disable()` para eliminar todos los mensajes que sean de ese nivel o inferior. Para deshabilitar todos los registros, añade `logging.disable(logging.CRITICAL)` a tu programa. Por ejemplo, escribe esto en el intérprete de comandos interactivo:

```
> > > import logging
```

```
> > > logging.basicConfig(level=logging.INFO, format='  
%(asctime)s -
```

```
%(levelname)s - %(message)s')
```

```
> > > logging.critical('Critical error! Critical error!')
```

```
2035-05-22 11:10:48,054 - CRITICAL - Critical error! Critical error!
```


> > > **logging.disable(logging.CRITICAL)**

> > > **logging.critical('Critical error! Critical error!')**

> > > **logging.error('Error! Error!')**

Dado que `logging.disable()` deshabilitará todos los mensajes de después, es probable que te convenga añadirlo cerca de la línea `import logging` de tu programa. De ese modo, puedes encontrarlo fácilmente para habilitar o deshabilitar con comentarios esa llamada y permitir o no permitir los mensajes de registro según sea necesario.

El depurador de Mu

El depurador es una función de Mu, IDLE y otros editores que permite ejecutar un programa línea por línea. Ejecuta una línea y espera a que le digamos que continúe. Al ejecutar un programa de esta forma, podemos tomarnos el tiempo que necesitemos para examinar los valores de las variables en cualquier punto durante la vida del programa. Es una herramienta valiosa para detectar fallos.

Para ejecutar un programa con el depurador de Mu, haz clic en el botón Depurar, en la parte superior de la pantalla, junto a Ejecutar. Debería abrirse el panel Inspector de depuración a la derecha de la ventana. Este panel muestra el valor actual de las variables del programa. En la figura 5.1, el depurador ha pausado la ejecución del programa justo antes de ejecutar la primera línea de código. Se ve esta línea resaltada en el editor de archivos.

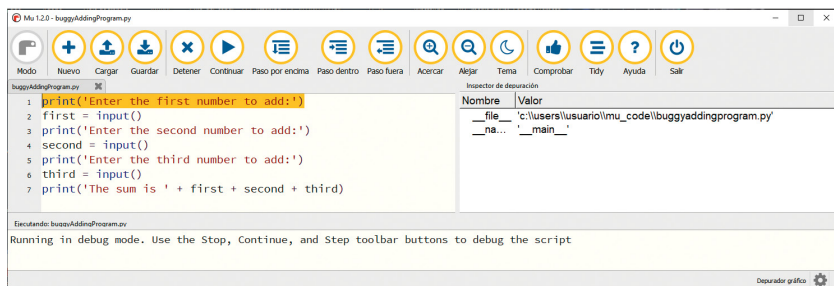


Figura 5.1. Mu ejecutando un programa con el depurador.

El modo de depuración también añade los siguientes botones en la parte superior del editor: Continuar, Paso por encima, Paso dentro y Paso fuera. El botón Detener habitual también está disponible.

Hacer clic en el botón Continuar hará que el programa se ejecute con normalidad hasta que termine o llegue a un punto de interrupción (hablaremos de estos puntos más adelante en el capítulo). Si has terminado de depurar y quieres que el programa continúe con normalidad, haz clic en Continuar.

Hacer clic en el botón Paso dentro hará que el depurador ejecute la siguiente línea de código y vuelva a detenerse. Si la siguiente línea es una llamada a una función, el depurador entrará en la función, saltando a su primera línea de código.

Hacer clic en el botón Paso por encima ejecutará la siguiente línea de código, como el botón Paso dentro. Sin embargo, si la siguiente línea es una llamada a una función, Paso por encima pasará muy deprisa por el código de la función. El código de la función se ejecutará a toda pastilla y el depurador se detendrá en cuanto regrese la llamada a la función. Por ejemplo, si la siguiente línea de código llama a una función `spam()`, pero en realidad no te interesa el código dentro de esa función, puedes hacer clic en Paso por encima para ejecutar el código de la función a velocidad normal y hacer la pausa cuando vuelva la función. Por eso es más habitual usar Paso por encima que Paso dentro.

Hacer clic en Paso fuera hará que el depurador ejecute líneas de código a toda velocidad hasta regresar de la función actual. Si has entrado en una llamada a función con el botón Paso dentro y ahora solo quieres que se sigan ejecutando instrucciones hasta que salgas, haz clic en Paso fuera para salir de la llamada a la función actual.

Si quieres dejar de depurar y no molestarte en seguir ejecutando el resto del programa, haz clic en Detener. Este botón terminará el programa inmediatamente.

Depurar un programa de suma

Para practicar con el depurador de Mu, abre una pestaña nueva en el editor de archivos y escribe el siguiente código:

```
print('Enter the first number to add:')  
  
first = input()  
  
print('Enter the second number to add:')  
  
second = input()  
  
print('Enter the third number to add:')  
  
third = input()  
  
print('The sum is ' + first + second + third)
```

Guarda el programa como `buggyAddingProgram.py` y ejecútalo primero sin el depurador. La salida será algo como esto:

Enter the first number to add:

5

Enter the second number to add:

3

Enter the third number to add:

42

The sum is 5342

El programa no ha fallado, pero, obviamente, la suma está mal.

Ejecuta el programa de nuevo, esta vez con el depurador. Haz clic en Depurar; el programa debería pararse en la línea 1, que es el código que está a punto de ejecutar.

Haz clic en Paso por encima una vez para ejecutar la primera llamada a `print()`. Usamos Paso por encima en vez de Paso dentro porque no queremos entrar en el código de la función `print()` (aunque Mu debería evitar que el depurador entre en las funciones integradas de Python). El depurador avanza a la línea 2 y la destaca en el editor de archivos, como muestra la figura 5.2. Así vemos

dónde se encuentra ahora mismo la ejecución del programa.

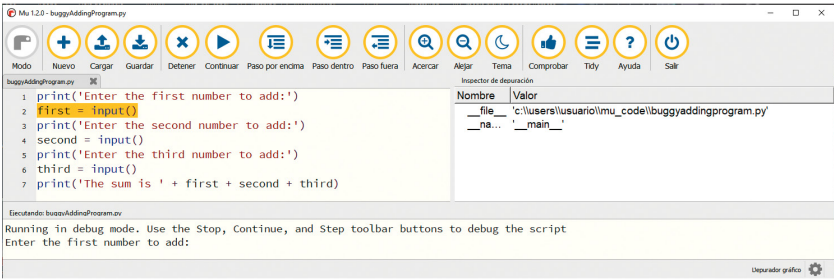


Figura 5.2. La ventana del editor Mu tras hacer clic en Paso por encima.

Haz clic en Paso por encima para ejecutar la llamada a la función `input()`. El resaltado desaparecerá mientras Mu espera a que escribas algo para la llamada a `input()` en el panel de salida. Escribe 5 y pulsa **Intro**. Volverá el resaltado.

Sigue haciendo clic en Paso por encima y escribe 3 y 42 como los dos siguientes números. Cuando el depurador llegue a la línea 7, la última llamada a `print()` del programa, la ventana del editor Mu debería ser como la figura 5.3.

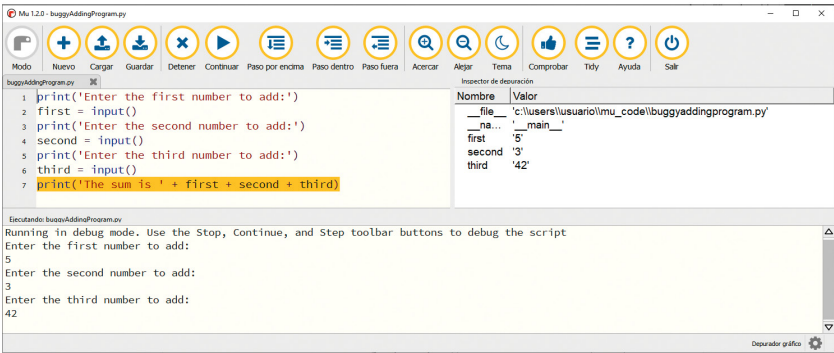


Figura 5.3. El panel Inspector de depuración, situado a la derecha de la ventana del editor Mu, muestra que las variables están establecidas como cadenas y no como enteros, provocando así el fallo.

En el panel Inspector de depuración, deberías ver que la primera, la

segunda y la tercera variables son valores de cadena: '5', '3' y '42', en lugar de los valores enteros 5, 3 y 42. Cuando se ejecuta la última línea, Python concatena estas cadenas en lugar de sumar los números, provocando el error.

Ir pasando por el programa con el depurador es útil, pero también puede ser lento. A menudo, interesa que el programa se ejecute con normalidad hasta que llegue a determinada línea de código. Podemos configurar el depurador para que haga eso con puntos de interrupción.

Configurar puntos de interrupción

Configurar un punto de interrupción en una línea de código específica obliga al depurador a parar siempre que la ejecución del programa llegue a esa línea. Abre una pestaña nueva en el editor de archivos e introduce el siguiente programa, que simula mil lanzamientos de moneda. Guárdalo como coinFlip.py:

```
import random

heads = 0

for i in range(1, 1001):

    1 if random.randint(0, 1) == 1:

        heads = heads + 1

    if i == 500:

        2 print('Halfway done!')

print('Heads came up ' + str(heads) + ' times.')
```

La llamada a `random.randint(0, 1)` 1 devolverá 0 la mitad de las veces y 1 la otra mitad, simulando un lanzamiento de moneda 50/50, donde 1 es cara. Al ejecutar el programa sin el depurador, enseguida producirá algo como esto:

Halfway done!

Heads came up 490 times.

Si ejecutásemos el programa con el depurador, tendríamos que hacer clic en Paso por encima miles de veces antes de que el programa terminase. Si nos interesase el valor de caras a mitad de la ejecución del programa, donde se han completado 500 de 1.000 lanzamientos, podríamos sencillamente establecer un punto de interrupción en la línea `print('Halfway done!')` 2. Para configurar un punto de interrupción, haz clic en el número de línea en el editor de archivos. Aparecerá un punto rojo señalando el punto de interrupción, como en la figura 5.4.

Observa que no nos interesa poner un punto de interrupción en la línea de la declaración `if`, ya que esta se ejecuta en todas las iteraciones por el bucle. Cuando establecemos el punto de interrupción en el código de la declaración `if`, el depurador para solo cuando la ejecución entra en la proposición `if`. Ahora, al ejecutar el programa con el depurador, debería empezar en pausa en la primera línea, como siempre, pero, al hacer clic en Continuar, debería ejecutarse a velocidad normal hasta alcanzar la línea que tiene el punto de interrupción. Entonces podemos hacer clic en Continuar, Paso por encima, Paso dentro o Paso fuera para continuar con normalidad.



Figura 5.4. Establecer un punto de interrupción coloca un punto rojo junto al número de línea.

Para quitar un punto de interrupción, hay que volver a hacer clic en el número de línea. El punto rojo desaparecerá y el depurador no se detendrá en esa línea en el futuro.

Resumen

Las aserciones, las excepciones, los registros y el depurador son herramientas valiosas para descubrir y evitar fallos en un programa. Las aserciones con la declaración `assert` de Python son una buena manera de implementar “pruebas de cordura” que te avisen pronto cuando una condición necesaria no se da. Las aserciones son solo para errores de los que el programa no debería intentar recuperarse y deberían fallar pronto. De lo contrario, habría que lanzar una excepción.

Una excepción se puede capturar y manejar con las declaraciones `try` y `except`. El módulo `logging` es una buena manera de inspeccionar el código mientras se ejecuta y es mucho más cómodo que usar la función `print()` por sus distintos niveles de registro y capacidad para guardar el registro en un archivo de texto.

El depurador permite pasar por el programa línea por línea. Como alternativa, se puede ejecutar el programa a velocidad normal y hacer que el depurador ponga en pausa la ejecución cuando llegue a un punto de interrupción establecido. Con el depurador, podemos ver el estado del valor de cualquier variable en cualquier momento durante la vida del programa.

Introducir por accidente fallos en el código es algo muy corriente, da igual cuántos años lleves programando. Estas herramientas y técnicas de depuración te ayudarán a escribir programas que funcionen.

Preguntas prácticas

1. Escribe una declaración `assert` que dé un error `AssertionError` si la variable `spam` es un entero menor que 10.
2. Escribe una declaración `assert` que dé un error `AssertionError` si las variables `eggs` y `bacon` contienen cadenas que sean iguales, incluso aunque el uso de mayúsculas y minúsculas sea distinto. (Es decir, ‘hello’ y ‘hello’ se consideran lo mismo, igual que ‘goodbye’ y ‘GOODbye’).
3. Escribe una declaración `assert` que siempre desencadene un error `AssertionError`.

4.¿Qué dos líneas debe tener un programa para poder llamar a logging.debug()?

5.¿Qué dos líneas debe tener un programa para hacer que logging.debug() envíe un mensaje de registro a un archivo llamado programLog.txt?

6.¿Cuáles son los cinco niveles de registro?

7.¿Qué línea de código podemos añadir para deshabilitar todos los mensajes de registro de un programa?

8.¿Por qué usar mensajes de registro es mejor que utilizar print() para mostrar el mismo mensaje?

9.¿Cuáles son las diferencias entre los botones Paso por encima, Paso dentro y Paso fuera del depurador?

10.¿Cuándo se detiene el depurador después de hacer clic en Continuar?

11.¿Qué es un punto de interrupción?

12.¿Cómo se establece un punto de interrupción en una línea de código de Mu?

Programa de práctica: depurar el lanzamiento de moneda

El siguiente programa está pensado para ser un sencillo juego de adivinación del resultado de lanzar una moneda. El jugador tiene dos intentos. (Es un juego fácil). Sin embargo, el programa tiene muchos fallos. Repásalo unas cuantas veces para descubrir los fallos que impiden que funcione correctamente.

```
import random
```

```
guess = “
```

```
while guess not in (‘heads’, ‘tails’):
```

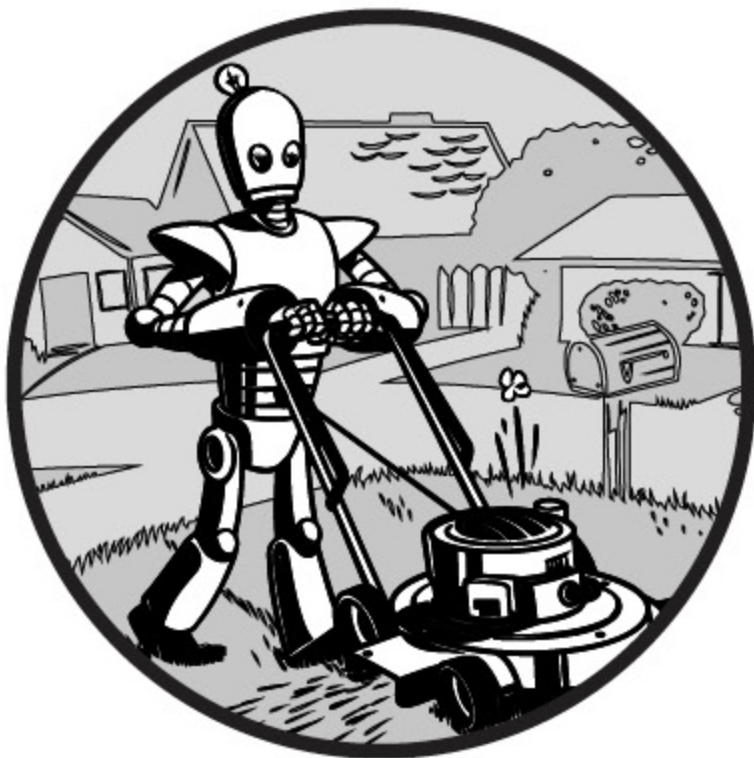
```
print(“Guess the coin toss! Enter heads or tails:”)
```



```
guess = input()

toss = random.randint(0, 1) # 0 es cruz, 1 es cara.

if toss == guess:
    print("You got it!")
else:
    print("Nope! Guess again!")
    guess = input()
    if toss == guess:
        print("You got it!")
    else:
        print("Nope. You are really bad at this game.")
```



Otro tema que debes entender antes de empezar a escribir programas en serio es el tipo de datos lista y su prima, la tupla. Las listas y las tuplas pueden contener múltiples valores, lo que hace que sea más fácil escribir programas que manejan grandes cantidades de datos. Y, dado que las listas pueden contener otras listas, puedes usarlas para organizar datos en estructuras jerárquicas.

En este capítulo, veremos lo básico de las listas. También te hablaré de los métodos, que son funciones vinculadas a los valores de determinados tipos de datos. Después, explicaré brevemente los

tipos de datos de secuencia (listas, tuplas y cadenas) y sus diferencias. En el próximo capítulo, te presentaré el tipo de datos diccionario.

El tipo de datos lista

Una lista es un valor que contiene varios valores en una secuencia ordenada. El término “valor de lista” se refiere a la propia lista (que puede almacenarse en una variable o pasarse a una función como cualquier otro valor), no a los valores dentro de la misma. Un valor de lista tiene este aspecto: `['cat', 'bat', 'rat', 'elephant']`. Igual que los valores de cadena utilizan las comillas simples para marcar dónde empieza y dónde acaba la cadena, la lista empieza y termina con corchetes, `[`.

Los valores que hay dentro de una lista se llaman elementos. Los elementos van separados por comas. Para ver un ejemplo, escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > [1, 2, 3] # Una lista de tres enteros
```

```
[1, 2, 3]
```

```
> > > ['cat', 'bat', 'rat', 'elephant'] # Una lista de cuatro cadenas
```

```
['cat', 'bat', 'rat', 'elephant']
```

```
> > > ['hello', 3.1415, True, None, 42] # Una lista de varios valores
```

```
['hello', 3.1415, True, None, 42]
```

```
1 > > > spam = ['cat', 'bat', 'rat', 'elephant']
```

```
> > > spam
```

```
['cat', 'bat', 'rat', 'elephant']
```

A la variable `spam` 1 se le asigna solo un valor: la lista. Pero la lista contiene a su vez otros valores.

Observa que el valor `[]` es una lista vacía, igual que `""` es una cadena vacía.

Índices

Supongamos que tenemos la lista `['cat', 'bat', 'rat', 'elephant']` almacenada en una variable llamada `spam`. El código de Python `spam[0]` se evaluaría como `'cat'`, `spam[1]` como `'bat'`, y así sucesivamente. El entero dentro de los corchetes que hay a continuación de la lista se denomina índice. El primer valor de la lista está en el índice 0, el segundo en el 1, el tercero en el 2, etc. La figura 6.1 muestra un valor de lista asignado a `spam`, junto con las expresiones de índice a las que se evalúan. Observa que, como el primer índice es 0, el último es el tamaño de la lista menos uno, así que una lista de cuatro elementos tendrá 3 como último índice.

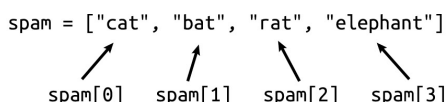


Figura 6.1. Un valor de lista almacenado en la variable `spam`, que muestra a qué valor se refiere cada índice.

Para ver un ejemplo de trabajo con índices, escribe las siguientes expresiones en el intérprete de comandos interactivo. Empezaremos asignando una lista a la variable `spam`:

```
> > > spam = ['cat', 'bat', 'rat', 'elephant']
```

```
> > > spam[0]
```

```
'cat'
```

```
> > > spam[1]
```

```
'bat'
```

```
> > > spam[2]
```

```
'rat'
```

```
> > > spam[3]
```

'elephant'

```
> > > ['cat', 'bat', 'rat', 'elephant'] [3]
```

1 'elephant'

```
2 > > > 'Hello, ' + spam[0]
```

'Hello, cat'

```
> > > 'The ' + spam[1] + ' ate the ' + spam[0] + '
```

'The bat ate the cat.'

Observa que la expresión 'Hello, ' + spam[0] se evalúa a 'Hello, ' + 'cat' porque spam[0] se evalúa como la cadena 'cat'. A su vez, esta expresión se evalúa como la cadena 'Hello, cat' 2.

Python muestra un mensaje de error `IndexError` si usamos un índice que supera el número de valores de la lista:

```
> > spam = ['cat', 'bat', 'rat', 'elephant']
```

```
> > > spam[10000]
```

Traceback (most recent call last):

File "<python-input-0>", line 1, in <module>

spam[10000]

`IndexError: list index out of range`

Las listas también pueden contener otras listas. Podemos acceder a los valores de estas listas utilizando varios índices, así:

```
> > > spam = [['cat', 'bat'], [10, 20, 30, 40, 50]]
```

```
> > > spam[0]
```

['cat', 'bat']

```
> > > spam[0][1]
```

'bat'

```
> > > spam[1][4]
```

50

El primer índice indica qué valor de lista usar y el segundo indica el valor dentro de la lista. Por ejemplo, `spam[0][1]` imprime 'bat', que es el segundo valor de la primera lista.

Índices negativos

Aunque los índices empiezan en 0 y van subiendo, también podemos utilizar enteros negativos para el índice. Por ejemplo, escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > spam = ['cat', 'bat', 'rat', 'elephant']
```

```
> > > spam[-1] # Último índice
```

'elephant'

```
> > > spam[-3] # Antepenúltimo índice
```

'bat'

```
> > > 'The ' + spam[-1] + ' is afraid of the ' + spam[-3] + '
```

'The elephant is afraid of the bat.'

El valor entero -1 se refiere al último índice de una lista, el valor -2 al penúltimo y así sucesivamente.

Rebanadas

Igual que un índice puede obtener un único valor de una lista, una rebanada puede obtener varios, en forma de una nueva lista. Las rebanadas van entre corchetes, como los índices, pero incluyen dos enteros separados por dos puntos. Observa la diferencia entre índices y rebanadas:

- spam[2 es una lista con un índice (un entero).

- spam[1:4 es una lista con una rebanada (dos enteros).

En una rebanada, el primer entero es el índice donde empieza la rebanada; el segundo índice es el entero donde termina. La lista creada a partir de una rebanada subirá hasta, pero no incluirá, el valor del segundo índice. Para ver un ejemplo, escribe esto en el intérprete de comandos interactivo:

```
> > > spam = ['cat', 'bat', 'rat', 'elephant']
```

```
> > > spam[0:4]
```

```
['cat', 'bat', 'rat', 'elephant']
```

```
> > > spam[1:3]
```

```
['bat', 'rat']
```

```
> > > spam[0:-1]
```

```
['cat', 'bat', 'rat']
```

Como atajo, puedes saltarte uno o los dos índices a ambos lados de los dos puntos de la rebanada:

```
> > > spam = ['cat', 'bat', 'rat', 'elephant']
```

```
> > > spam[:2]
```

```
['cat', 'bat']
```

```
> > > spam[1:]
```

```
['bat', 'rat', 'elephant']
```

```
> > > spam[:]
```

```
['cat', 'bat', 'rat', 'elephant']
```

Saltarse el primer índice es como usar 0 o el principio de la lista. Saltarse el segundo índice es como usar el tamaño de la lista, la

rebanada llegará hasta el final.

La función len()

La función len() devolverá el número de valores de una lista que se le pase. Por ejemplo, escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > spam = ['cat', 'dog', 'moose']
```

```
> > > len(spam)
```

3

Este comportamiento es similar a cómo la función cuenta el número de caracteres en un valor de cadena.

Actualización de valores

Por lo general, el nombre de una variable va a la izquierda de una declaración de asignación, como en `spam = 42`. Sin embargo, también se puede usar un índice de una lista para cambiar el valor de ese índice:

```
> > > spam = ['cat', 'bat', 'rat', 'elephant']
```

```
> > > spam[1] = 'aardvark'
```

```
> > > spam
```

```
['cat', 'aardvark', 'rat', 'elephant']
```

```
> > > spam[2] = spam[1]
```

```
> > > spam
```

```
['cat', 'aardvark', 'aardvark', 'elephant']
```

```
> > > spam[-1] = 12345
```

```
> > > spam
```



```
['cat', 'aardvark', 'aardvark', 12345]
```

En este ejemplo, `spam[1] = 'aardvark'` significa “asigna el valor del índice 1 de la lista `spam` a la cadena ‘aardvark’”. También podemos usar índices negativos como -1 para actualizar listas.

Concatenación y replicación

Podemos concatenar y replicar listas con los operadores `+` y `*`, igual que las cadenas:

```
> > > [1, 2, 3] + ['A', 'B', 'C']
```

```
[1, 2, 3, 'A', 'B', 'C']
```

```
> > > ['X', 'Y', 'Z'] * 3
```

```
['X', 'Y', 'Z', 'X', 'Y', 'Z', 'X', 'Y', 'Z']
```

```
> > > spam = [1, 2, 3]
```

```
> > > spam = spam + ['A', 'B', 'C']
```

```
> > > spam
```

```
[1, 2, 3, 'A', 'B', 'C']
```

El operador `+` combina dos listas para crear un nuevo valor de lista y el operador `*` combina una lista y un valor entero para replicar la lista.

Declaraciones del

La declaración `del` eliminará los valores de un índice de una lista. Todos los valores que queden detrás de un valor eliminado cambiarán de índice. Por ejemplo, escribe este en el intérprete de comandos interactivo:

```
> > > spam = ['cat', 'bat', 'rat', 'elephant']
```

```
> > > del spam[2]
```

```
> > > spam
```

```
['cat', 'bat', 'elephant']
```

```
> > > del spam [2]
```

```
> > > spam
```

```
['cat', 'bat']
```

La declaración `del` también puede funcionar con una simple variable para borrarla, como si fuese una declaración de “desasignación”. Si intentamos usar una variable después de eliminarla, obtendremos un error `NameError` porque esa variable ya no existe. En la práctica, casi nunca hace falta borrar variables simples, pero la declaración `del` es útil para eliminar valores de listas.

Trabajar con listas

Cuando empiezas a escribir programas, puedes caer en la tentación de crear varias variables individuales para almacenar un grupo de valores similares. Por ejemplo, si quisiese almacenar los nombres de mis gatos, podría pensar en un código así:

```
cat_name_1 = 'Zophie'
```

```
cat_name_2 = 'Pooka'
```

```
cat_name_3 = 'Simon'
```

```
cat_name_4 = 'Lady Macbeth'
```

Pero resulta que no es una buena manera de escribir código por una razón: si el número de gatos cambia (siempre se puede tener más gatos), el programa nunca será capaz de almacenar más gatos de los que tienes en las variables. Además, un programa así contendría mucho código duplicado o casi idéntico. Para ver esto en la práctica, escribe el siguiente programa en el editor de archivos y guárdalo como `allMyCats1.py`:

```
print('Enter the name of cat 1:')
```

```

cat_name_1 = input()

print('Enter the name of cat 2:')

cat_name_2 = input()

print('Enter the name of cat 3:')

cat_name_3 = input()

print('Enter the name of cat 4:')

cat_name_4 = input()

print('The cat names are:')

print(cat_name_1 + ' ' + cat_name_2 + ' ' + cat_name_3 + ' ' +
cat_name_4)

```

En lugar de utilizar múltiples variables repetitivas, podemos usar una sola variable que contenga un valor de lista. Como ejemplo, aquí tenemos una nueva versión mejorada del programa allMyCats1.py. Esta nueva versión emplea una única lista y puede almacenar cualquier número de gatos que introduzca el usuario. En una nueva ventana del editor de archivos, escribe el siguiente código fuente y guárdalo como allMyCats2.py:

```

cat_names = []

while True:

    print('Enter the name of cat ' + str(len(cat_names) + 1) +
    ' (Or enter nothing to stop.):')

    name = input()

    if name == '':

        break

    cat_names = cat_names + [name] # Concatenación de listas

```

```
print("The cat names are:")
```

```
for name in cat_names:
```

```
    print(' ' + name)
```

Cuando ejecutes el programa, la salida será algo como esto:

Enter the name of cat 1 (Or enter nothing to stop.):

Zophie

Enter the name of cat 2 (Or enter nothing to stop.):

Pooka

Enter the name of cat 3 (Or enter nothing to stop.):

Simon

Enter the name of cat 4 (Or enter nothing to stop.):

Lady Macbeth

Enter the name of cat 5 (Or enter nothing to stop.):

The cat names are:

Zophie

Pooka

Simon

Lady Macbeth

La ventaja de usar una lista es que los datos están estructurados, así que el programa puede procesarlos con mucha más flexibilidad que si hubiese variables repetitivas.

Bucles for y listas

En el capítulo 3, aprendiste a usar bucles for para ejecutar un

bloque de código una cantidad determinada de veces.

Técnicamente, el bucle for repite el bloque de código una vez por cada valor de una lista. Por ejemplo, si ejecutamos este código

```
for i in range(4):
```

```
    print(i)
```

la salida del programa será así:

```
0
```

```
1
```

```
2
```

```
3
```

Esto se debe a que el valor de retorno de range(4) es un valor de secuencia que Python considera similar a [0, 1, 2, 3]. El siguiente programa tiene la misma salida que el anterior:

```
for i in [0, 1, 2, 3]:
```

```
    print(i)
```

El bucle for anterior pasa en realidad por su proposición con la variable i establecida en un valor sucesivo en la lista [0, 1, 2, 3] en cada iteración.

Una técnica habitual de Python es usar range(len(some_list)) con un bucle for para iterar por los índices de una lista. Por ejemplo, introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > supplies = ['pens', 'staplers', 'flamethrowers',  
'binders']
```

```
> > > for i in range(len(supplies)):
```

```
...     print('Index ' + str(i) + ' in supplies is: ' + supplies[i])
```

```
...
```

Index 0 in supplies is: pens

Index 1 in supplies is: staplers

Index 2 in supplies is: flamethrowers

Index 3 in supplies is: binders

Utilizar `range(len(supplies))` en el bucle `for` anterior es útil porque el código del bucle puede acceder al índice (como la variable `i`) y el valor de ese índice (como `supplies[i]`). Y lo mejor de todo es que `range(len(supplies))` iterará por todos los índices de `supplies`, independientemente de cuántos elementos contenga la lista.

Los operadores `in` y `not in`

Podemos determinar si un valor está o no está en una lista con los operadores `in` y `not in`. Al igual que otros operadores, `in` y `not in` van en expresiones y contienen dos valores: uno que buscar en una lista y la lista donde puede estar. Estas expresiones se evaluarán como un valor booleano. Para ver cómo funciona esto, escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > 'howdy' in ['hello', 'hi', 'howdy', 'heyas']
```

True

```
> > > spam = ['hello', 'hi', 'howdy', 'heyas']
```

```
> > > 'cat' in spam
```

False

```
> > > 'howdy' not in spam
```

False

```
> > > 'cat' not in spam
```

True

El siguiente programa permite al usuario introducir un nombre de

mascota y luego comprueba si ese nombre está en una lista de mascotas. Abre una nueva ventana en el editor de archivos, introduce el siguiente código y guárdalo como myPets.py:

```
my_pets = ['Zophie', 'Pooka', 'Fat-tail']

print('Enter a pet name:')

name = input()

if name not in my_pets:

    print('I do not have a pet named ' + name)

else:

    print(name + ' is my pet.')
```

La salida podría ser algo así:

Enter a pet name:

Footfoot

I do not have a pet named Footfoot

Ten en cuenta que el operador not in es distinto del operador booleano not.

El truco de la asignación múltiple

El truco de la asignación múltiple (llamado técnicamente “desempaquetado de tuplas”) es un atajo que permite asignar múltiples variables con los valores de una lista en una línea de código. De ese modo, en vez de hacer esto:

```
> > > cat = ['fat', 'gray', 'loud']

> > > size = cat[0]

> > > color = cat[1]
```

```
> > > disposition = cat[2]
```

Podemos hacer esto:

```
> > > cat = ['fat', 'gray', 'loud']
```

```
> > > size, color, disposition = cat
```

La cantidad de variables y la longitud de la lista debe ser exactamente igual, de lo contrario, Python dará un error `ValueError`:

```
> > > cat = ['fat', 'gray', 'loud']
```

```
> > > size, color, disposition, name = cat
```

Traceback (most recent call last):

File "<python-input-0>", line 1, in <module>

`size, color, disposition, name = cat`

`ValueError: not enough values to unpack (expected 4, got 3)`

Este truco hace que nuestro código sea más corto y más legible que si escribiésemos tres líneas separadas.

Enumeración de elementos de una lista

En lugar de usar la técnica `range(len(some_list))` con un bucle `for` para obtener el índice entero de los elementos de la lista, podemos llamar a la función `enumerate()`. En cada iteración del bucle, `enumerate()` devolverá dos valores: el índice del elemento de la lista y el elemento en sí. Por ejemplo, este código equivale al del apartado "Bucles `for` y listas":

```
> > > supplies = ['pens', 'staplers', 'flamethrowers',  
               'binders']
```

```
> > > for index, item in enumerate(supplies):
```

```
... print('Index ' + str(index) + ' in supplies is: ' + item)
```


...

Index 0 in supplies is: pens

Index 1 in supplies is: staplers

Index 2 in supplies is: flamethrowers

Index 3 in supplies is: binders

La función `enumerate()` es útil si necesitas tanto el elemento como su índice en el bloque del bucle.

Selección y orden aleatorios

El módulo `random` tiene un par de funciones que aceptan listas como argumentos. La función `random.choice()` devolverá un elemento seleccionado al azar de la lista. Escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > import random
```

```
> > > pets = ['Dog', 'Cat', 'Moose']
```

```
> > > random.choice(pets)
```

```
'Cat'
```

```
> > > random.choice(pets)
```

```
'Cat'
```

```
> > > random.choice(pets)
```

```
'Dog'
```

Podemos considerar `random.choice(some_list)` como una forma abreviada de `some_list[random.randint(0, len(some_list) - 1)`.

La función `random.shuffle()` reordenará los elementos de una lista dada. Escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > import random

> > > people = ['Alice', 'Bob', 'Carol', 'David']

> > > random.shuffle(people)

> > > people

['Carol', 'David', 'Alice', 'Bob']

> > > random.shuffle(people)

> > > people

['Alice', 'David', 'Bob', 'Carol']
```

Esta función modifica la lista en cuestión en lugar de devolver una lista nueva.

Operadores de asignación aumentada

Los operadores `+` y `*` que funcionan con las cadenas también funcionan con las listas, así que vamos a dar un pequeño rodeo para hablar de los operadores de asignación aumentada. Al asignar un valor a una variable, normalmente usamos la propia variable. Por ejemplo, después de asignar 42 a la variable `spam`, aumentaríamos el valor de `spam` en 1 con el siguiente código:

```
> > > spam = 42

> > > spam = spam + 1

> > > spam
```

43

Como atajo, podemos usar el operador de asignación aumentada `+=` (que es el operador normal seguido de un signo de igualdad) para hacer lo mismo:

```
> > > spam = 42
```

```
> > > spam = 1
```

```
> > > spam
```

43

Hay operadores de asignación aumentada para los operadores +, -, *, / y %, están descritos en la tabla 6.1.

Tabla 6.1. Los operadores de asignación aumentada.

Declaración de asignación aumentada

~~spam += spam + 1~~

~~spam -= spam - 1~~

~~spam *= spam * 1~~

~~spam /= spam / 1~~

~~spam %= spam % 1~~

El operador += también puede concatenar cadenas y listas y el operador *= puede replicar cadenas y listas. Escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > spam = 'Hello,'
```

```
> > > spam = ' world!' # Igual que spam = spam + 'world!'
```

```
> > > spam
```

```
'Hello, world!'
```

```
> > > bacon = ['Zophie']
```

```
> > > bacon = 3 # Igual que bacon = bacon * 3
```

```
> > > bacon
```

```
['Zophie', 'Zophie', 'Zophie']
```

Al igual que el truco de la asignación múltiple, los operadores de asignación aumentada son un atajo para que el código sea más sencillo y legible.

Métodos

Un método es lo mismo que una función, salvo por que se llama sobre un valor. Por ejemplo, si hubiese un valor de lista almacenado en `spam`, llamaríamos al método de lista `index()` (que explicaré enseguida) sobre esa lista: `spam.index('hello')`. La parte del método va después del valor, separada por un punto. Cada tipo de datos tiene su propio conjunto de métodos. El tipo de datos lista, por ejemplo, cuenta con varios métodos útiles para buscar, añadir, eliminar y manipular de otras maneras valores de una lista. Piensa en un método como en una función que va siempre asociada a un valor. En nuestro ejemplo de la lista `spam`, la función sería hipotéticamente `index(spam, 'hello')`. Pero, como `index()` es un método de lista y no una función, llamamos a `spam.index('hello')`. Llamar a `index()` sobre un valor de lista es como Python sabe que `index()` es un método de lista. Vamos a ver los métodos de lista de Python.

Encontrar valores

Los valores de lista tienen un método `index()` al que se puede pasar un valor. Si ese valor existe en la lista, el método devolverá el índice del valor. Si el valor no está en la lista, Python produce un error `ValueError`. Escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > spam = ['hello', 'hi', 'howdy', 'heyas']
```

```
> > > spam.index('hello')
```

```
0
```

```
> > > spam.index('heyas')
```

```
3
```

```
> > > spam.index('howdy howdy howdy')
```

Traceback (most recent call last):

File "<python-input-0>", line 1, in <module>

```
spam.index('howdy howdy howdy')
```

ValueError: 'howdy howdy howdy' is not in list

Cuando la lista contiene duplicados del valor, el método devuelve el índice de la primera aparición:

```
> > > spam = ['Zophie', 'Pooka', 'Fat-tail', 'Pooka']
```

```
> > > spam.index('Pooka')
```

```
1
```

Observa que index() devuelve 1, no 3.

Añadir valores

Para añadir valores nuevos a una lista, utiliza los métodos append() e insert(). El método append() añade el argumento al final de la lista:

```
> > > spam = ['cat', 'dog', 'bat']
```

```
> > > spam.append('moose')
```

```
> > > spam
```

```
['cat', 'dog', 'bat', 'moose']
```

El método insert() puede insertar un valor en cualquier índice de la lista. El primer argumento para insert() es el índice del nuevo valor y el segundo, el valor que hay que insertar. Escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > spam = ['cat', 'dog', 'bat']
```

```
> > > spam.insert(1, 'chicken')
```

```
> > > spam
```

```
['cat', 'chicken', 'dog', 'bat']
```

Observa que el código no realiza ninguna operación de asignación, como `spam = spam.append('moose')` o `spam = spam.insert(1, 'chicken')`. El valor de retorno de `append()` e `insert()` es `None`, así que decididamente no interesa guardarlo como el nuevo valor de la variable. En vez de eso, estos métodos modifican la lista *in situ*, un tema del que hablaremos con más detalle en el apartado “Tipos de datos mutables e inmutables”. Los métodos pertenecen a un único tipo de datos. Los métodos `append()` e `insert()` son métodos de lista y los llamamos solo con valores de lista, no con valores de ningún otro tipo de datos, como cadenas o enteros. Para ver lo que pasa cuando intentamos hacer eso, escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > eggs = 'hello'
```

```
> > > eggs.append('world')
```

Traceback (most recent call last):

File “<python-input-0>”, line 1, in <module>

```
eggs.append('world')
```

AttributeError: 'str' object has no attribute 'append'

```
> > > bacon = 42
```

```
> > > bacon.insert(1, 'world')
```

Traceback (most recent call last):

File “<python-input-0>”, line 1, in <module>

```
bacon.insert(1, 'world')
```

AttributeError: 'int' object has no attribute 'insert'

Observa el mensaje de error `AttributeError` que aparece.

Quitar valores

El método `remove()` acepta un valor para quitar de la lista con la

que se le llama:

```
> > > spam = ['cat', 'bat', 'rat', 'elephant']
```

```
> > > spam.remove('bat')
```

```
> > > spam
```

```
['cat', 'rat', 'elephant']
```

Intentar borrar un valor que no existe en la lista producirá un error `ValueError`. Para ver un ejemplo, escribe lo siguiente en el intérprete de comandos interactivo y observa el error que muestra:

```
> > > spam = ['cat', 'bat', 'rat', 'elephant']
```

```
> > > spam.remove('chicken')
```

Traceback (most recent call last):

File "<python-input-0>", line 1, in <module>

```
spam.remove('chicken')
```

```
ValueError: list.remove(x): x not in list
```

Si el valor aparece varias veces en la lista, el método solo eliminará la primera aparición:

```
> > > spam = ['cat', 'bat', 'rat', 'cat', 'hat', 'cat']
```

```
> > > spam.remove('cat')
```

```
> > > spam
```

```
['bat', 'rat', 'cat', 'hat', 'cat']
```

La función `del` es útil cuando conoces el índice del valor que quieres eliminar, mientras que el método `remove()` lo es cuando conoces el valor en sí.

Almacenar valores

Podemos almacenar listas de valores numéricos o de cadenas con el método `sort()`. Por ejemplo, escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > spam = [2, 5, 3.14, 1, -7]
```

```
> > > spam.sort()
```

```
> > > spam
```

```
[-7, 1, 2, 3.14, 5]
```

```
> > > spam = ['Ants', 'Cats', 'Dogs', 'Badgers', 'Elephants']
```

```
> > > spam.sort()
```

```
> > > spam
```

```
['Ants', 'Badgers', 'Cats', 'Dogs', 'Elephants']
```

El método devuelve los números en orden numérico y las cadenas en orden alfabético. También se puede pasar `True` como argumento de la palabra clave `reverse` para ordenar los valores a la inversa:

```
> > > spam.sort(reverse=True)
```

```
> > > spam
```

```
['Elephants', 'Dogs', 'Cats', 'Badgers', 'Ants']
```

Hay tres cosas sobre el método `sort()` que debes saber. La primera es que ordena la lista *in situ*; no intentes capturar el valor de retorno escribiendo código como `spam = spam.sort()`.

La segunda es que no podemos ordenar listas que tienen tanto valores numéricos como cadenas porque Python no sabe cómo comparar esos valores. Escribe lo siguiente en el intérprete de comandos interactivo y observa el error `TypeError`:

```
> > > spam = [1, 3, 2, 4, 'Alice', 'Bob']
```

```
> > > spam.sort()
```


Traceback (most recent call last):

File "<python-input-0>", line 1, in <module>

spam.sort()

TypeError: '<' not supported between instances of 'str' and 'int'

Por último, `sort()` utiliza el orden alfabético de ASCII, no el orden alfabético normal que usa para las cadenas. Esto significa que las mayúsculas van antes que las minúsculas, colocando la `a` minúscula detrás de la `Z` mayúscula. Para ver un ejemplo, introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > spam = ['Alice', 'ants', 'Bob', 'badgers', 'Carol', 'cats']
```

```
> > > spam.sort()
```

```
> > > spam
```

```
['Alice', 'Bob', 'Carol', 'ants', 'badgers', 'cats']
```

Si necesitas ordenar los valores en orden alfabético normal, pasa `str.lower` como argumento de la palabra clave `key` en la llamada al método `sort()`:

```
> > > spam = ['a', 'z', 'A', 'Z']
```

```
> > > spam.sort(key=str.lower)
```

```
> > > spam
```

```
['a', 'A', 'z', 'Z']
```

Este argumento hace que `sort()` trate todos los elementos de la lista como si estuvieran en minúsculas, sin cambiar realmente los valores en la lista.

Invertir valores

Si necesitas invertir rápidamente el orden de los elementos de una lista, puedes llamar al método `reverse()`. Escribe esto en el

intérprete de comandos interactivo:

```
> > > spam = ['cat', 'dog', 'moose']
```

```
> > > spam.reverse()
```

```
> > > spam
```

```
['moose', 'dog', 'cat']
```

Al igual que el método `sort()`, `reverse()` no devuelve una lista, por eso escribimos `spam.reverse()` en vez de `spam = spam.reverse()`.

Excepciones a las reglas de sangrado en Python

En la mayoría de los casos, la cantidad de sangrado para una línea de código dice a Python en qué bloque está, pero hay unas pocas excepciones a esta regla. Por ejemplo, las listas pueden ocupar varias líneas en el archivo del código fuente y el sangrado de esas líneas no importa; Python sabe que la lista no termina hasta que encuentra el corchete de cierre. Esto significa que podemos escribir código como este:

```
pam = ['apples',
```

```
'oranges',
```

```
'bananas',
```

```
'cats']
```

```
print(spam[0]) # Imprime apples.
```

Naturalmente, hablando en un sentido práctico, la mayoría de la gente usa el comportamiento de Python para hacer que sus listas queden más bonitas y legibles, como la lista de mensajes del apartado “Programa corto: *Magic 8 Ball* con una lista”.

También podemos dividir una sola instrucción en varias líneas terminando cada una con el carácter de continuación de línea (`\`). Usar `\` es como decir: “Esta instrucción continúa en la línea siguiente”. El sangrado después de una continuación de línea no es

importante. Por ejemplo, esto es código Python válido:

```
print('Four score and seven ' + \
'years ago...')
```

Estos trucos son útiles cuando queremos reorganizar líneas largas de código Python para que sean más legibles.

Cortocircuito lógico con operadores booleanos

Los operadores booleanos tienen un comportamiento sutil que es fácil pasar por alto. Recuerda que, si uno de los dos valores combinados por un operador `and` es `False`, toda la expresión es `False` y, si cualquier valor combinado por un operador `or` es `True`, toda la expresión es `True`. Si te doy la expresión `False and spam`, da igual si la variable de `spam` es `True` o `False` porque toda la expresión será `False` en cualquier caso. Lo mismo para `True or spam`; esto se evalúa como `True` independientemente del valor de `spam`. Python (y otros muchos lenguajes de programación) usan esto para optimizar el código de manera que se ejecute un poco más deprisa al no examinar la parte derecha del operador booleano. Esto se denomina “cortocircuito lógico”. La mayoría de las veces, el programa se comportará igual que si Python comprobase la expresión entera (pero unos microsegundos más rápido). No obstante, fíjate en este programilla, donde comprobamos si el primer elemento de una lista es `'cat'`:

```
spam = ['cat', 'dog']

if spam[0] == 'cat':
    print('A cat is the first item.')
else:
    print('The first item is not a cat.')
```

Tal y como está escrito, este programa imprime `A cat is the first item`. Pero, si la lista de `spam` está vacía, el código `spam[0]` provocará un error `IndexError: list Index out of range`. Para

arreglarlo, vamos a ajustar la condición de la declaración if para aprovechar el cortocircuito:

```
spam = []

if len(spam) > 0 and spam[0] == 'cat':

    print('A cat is the first item.')

else:

    print('The first item is not a cat.')
```

Este programa nunca da error porque `if len(spam) > 0` es `False` (es decir, la lista de `spam` está vacía); el cortocircuito con el operador `and` hace que Python no se moleste en ejecutar el código `spam[0] == 'cat'` que daría el error `IndexError`. Ten en cuenta este comportamiento cuando escribas código que implique los operadores `and` y `or`.

Un programa corto: Magic 8 Ball con una lista

Usando listas, podemos escribir una versión mucho más elegante del programa del capítulo 4 `magic8Ball.py`. En lugar de crear varias líneas de declaraciones `elif` casi idénticas, podemos crear una sola lista con la que trabaje el código. Abre una nueva ventana en el editor de archivos y escribe el siguiente código. Guarda el programa como `magic8Ball2.py`.

```
import random

messages = ['It is certain',
            'It is decidedly so',
            'Yes definitely',
            'Reply hazy try again',
            'Ask again later',
            'Concentrate and ask again',
```

```
'My reply is no',  
'Outlook not so good',  
'Very doubtful']  
  
print('Ask a yes or no question:')  
  
input('>')  
  
print(messages[random.randint(0, len(messages) - 1)])
```

Al ejecutarlo, verás que funciona igual que el anterior programa `magic8Ball.py`. La llamada `random.randint(0, len(messages) - 1)` produce un número aleatorio que usar para el índice, al margen del tamaño de `messages`. Es decir, se obtiene un número aleatorio entre 0 y el valor de `len(messages) - 1`. La ventaja de este enfoque es que puedes añadir y quitar cadenas fácilmente en la lista de mensajes sin cambiar otras líneas de código. Si más adelante actualizas el código, tendrás que cambiar menos líneas, reduciendo las probabilidades de cometer fallos.

Seleccionar un elemento aleatorio de una lista es lo bastante habitual como para que Python tenga la función `random.choice(messages)`, que hace lo mismo que `random.randint(0, len(messages) - 1)`.

Tipo de datos de secuencia

Las listas no son los únicos tipos de datos que representan secuencias ordenadas de valores. Por ejemplo, las cadenas y las listas son muy similares si piensas en una cadena como en una “lista” de caracteres de texto. Los tipos de datos de secuencia de Python incluyen listas, cadenas, objetos de rango devueltos por `range()` y tuplas (de las que hablaremos enseguida). Muchas de las cosas que podemos hacer con listas pueden hacerse también con cadenas y otros valores de secuencias. Para verlo, escribe esto en el intérprete de comandos interactivo:

```
> > > name = 'Zophie'
```

```
> > > name[0]
```

```
'Z'
```

```
> > > name[-2]
```

```
'i'
```

```
> > > name[0:4]
```

```
'Zoph'
```

```
> > > 'Zo' in name
```

```
True
```

```
> > > 'z' in name
```

```
False
```

```
> > > 'p' not in name
```

```
False
```

```
> > > for i in name:
```

```
... print('* * * ' + i + ' * * ')
```

```
...
```

```
* * * Z * * *
```

```
* * * o * * *
```

```
* * * p * * *
```

```
* * * h * * *
```

```
* * * i * * *
```

```
* * * e * * *
```

Con los valores de secuencia se puede hacer lo mismo que con las

listas: índices, rebanadas, bucles for, usar len() y emplear los operadores in y not in.

Tipos de datos mutables e inmutables

Pese a todo, hay una diferencia sustancial entre listas y cadenas. Un valor de lista es un tipo de dato mutable: se puede añadir, eliminar y cambiar sus valores. En cambio, una cadena es inmutable: no se puede cambiar. Intentar reasignar un solo carácter de una cadena provocaría un error `TypeError`, como verás si pones esto en el intérprete de comandos interactivo:

```
> > > name = 'Zophie a cat'
```

```
> > > name[7] = 'the'
```

Traceback (most recent call last):

File "<python-input-0>", line 1, in <module>

```
name[7] = 'the'
```

`TypeError: 'str' object does not support item assignment`

La forma correcta de “mutar” una cadena es usar rebanadas y concatenación para construir una nueva cadena copiando partes de la antigua:

```
> > > name = 'Zophie a cat'
```

```
> > > new_name = name[0:7] + 'the' + name[8:12]
```

```
> > > name
```

```
'Zophie a cat'
```

```
> > > new_name
```

```
'Zophie the cat'
```

Hemos utilizado `[0:7]` y `[8:12]` para referirnos a los caracteres que no queremos sustituir. Observa que la cadena original 'Zophie a cat' no

se modifica porque las cadenas son inmutables. Aunque el valor de una lista es mutable, la segunda línea del siguiente código no modifica la lista `eggs`:

```
> > > eggs = ['A', 'B', 'C']
```

```
> > > eggs = ['x', 'y', 'z']
```

```
> > > eggs
```

```
['x', 'y', 'z']
```

El valor de lista de `eggs` no se cambia aquí, sino que un valor de lista nuevo y totalmente diferente (['x', 'y', 'z']) sustituye al antiguo valor de lista (['A', 'B', 'C']).

Si quisiéramos realmente modificar la lista original de `eggs` para que contenga ['x', 'y', 'z'], habría que usar la declaración del y el método `append()`, así:

```
> > > eggs = ['A', 'B', 'C']
```

```
> > > del eggs[2]
```

```
> > > del eggs[1]
```

```
> > > del eggs[0]
```

```
> > > eggs.append('x')
```

```
> > > eggs.append('y')
```

```
> > > eggs.append('z')
```

```
> > > eggs
```

```
['x', 'y', 'z']
```

En este ejemplo, la variable `eggs` termina con el mismo valor de lista con el que empezó. Es solo que esta lista se ha cambiado (mutado) en vez de sobrescrito. Esto se llama cambiar la lista *in situ*.

Diferenciar entre tipos mutables e inmutables puede parecer una

tontería, pero en “Referencias” explicaré la diferencia de comportamiento cuando se llama a funciones con argumentos mutables y cuando son inmutables. Pero antes, vamos a ver el tipo de datos tupla, que es una forma inmutable del tipo de datos lista.

El tipo de datos tupla

Solo hay dos diferencias entre los tipos de datos tupla y lista. La primera es que escribimos las tuplas con paréntesis en vez de corchetes. Escribe esto en el intérprete de comandos interactivo:

```
> > > eggs = ('hello', 42, 0.5)
```

```
> > > eggs[0]
```

```
'hello'
```

```
> > > eggs[1:3]
```

```
(42, 0.5)
```

```
> > > len(eggs)
```

```
3
```

La segunda, y más importante, diferencia entre las tuplas y las listas es que las tuplas, al igual que las cadenas, son inmutables: no se puede modificar, añadir ni eliminar sus valores. Escribe lo siguiente en el intérprete de comandos interactivo y mira el mensaje de error `TypeError` que da:

```
> > > eggs = ('hello', 42, 0.5)
```

```
> > > eggs[1] = 99
```

Traceback (most recent call last):

File “<python-input-0>”, line 1, in <module>

```
eggs[1] = 99
```

`TypeError: 'tuple' object does not support item assignment`

Si solo hay un valor en la tupla, podemos indicarlo poniendo una coma final después del valor entre paréntesis. Si no lo hacemos, Python pensará que hemos introducido un valor entre paréntesis normales. (A diferencia de lo que ocurre con otros lenguajes de programación, en Python está bien poner una coma final después del último elemento de una lista o tupla). Escribe las siguientes llamadas a la función `type()` en el intérprete de comandos interactivo para ver la diferencia:

```
> > > type(('hello',))
```

```
<class 'tuple'>
```

```
> > > type(('hello'))
```

```
<class 'str'>
```

Podemos usar tuplas para transmitir a cualquiera que lea nuestro código que no queremos que esos valores cambien. Si necesitas una secuencia ordenada de valores que no cambie nunca, utiliza una tupla. Una segunda ventaja de usar tuplas en vez de listas es que, como son inmutables y su contenido no cambia, Python puede implementar optimizaciones que hacen que un código que emplea tuplas sea un poco más rápido que el que se sirve de listas.

Conversión entre los tipos lista y tupla

Igual que `str(42)` devolverá `'42'`, la representación de la cadena del entero 42, las funciones `list()` y `tuple()` devolverán versiones en lista y tupla de los valores que se les pasen. Escribe lo siguiente en el intérprete de comandos interactivo y observa que el valor de retorno es de un tipo de datos diferente que el valor pasado:

```
> > > tuple(['cat', 'dog', 5])
```

```
('cat', 'dog', 5)
```

```
> > > list(('cat', 'dog', 5))
```

```
['cat', 'dog', 5]
```

```
> > > list('hello')
```

```
['h', 'e', 'l', 'l', 'o']
```

Convertir una tupla en una lista es útil cuando se necesita una versión mutable de un valor de tupla.

Referencias

Una metáfora habitual es que las variables son cajas que “guardan” valores como cadenas y enteros. Sin embargo, esta explicación es una simplificación de lo que Python está haciendo en realidad. Una metáfora mejor es que las variables son etiquetas unidas a valores con una cuerdecilla. Escribe lo siguiente en el intérprete de comandos interactivo:

```
1 > > > spam = 422
```

```
2 > > > eggs = spam
```

```
3 > > > spam = 99
```

```
> > > spam
```

```
99
```

```
> > > eggs
```

```
42
```

Cuando asignamos 42 a la variable `spam`, en realidad estamos creando el valor 42 en la memoria del ordenador y guardando la referencia al mismo en la variable `spam`. Cuando copiamos el valor de `spam` y se lo asignamos a la variable `eggs`, estamos copiando la referencia. Tanto la variable `spam` como la variable `eggs` hacen referencia al valor 42 de la memoria del ordenador. Siguiendo con la metáfora de la etiqueta para las variables, hemos puesto las `spam` y `eggs` al mismo valor 42. Cuando asignamos a `spam` un nuevo valor 99, cambiamos a lo que hace referencia la etiqueta `spam`. La figura 6.2 es una representación gráfica del código.

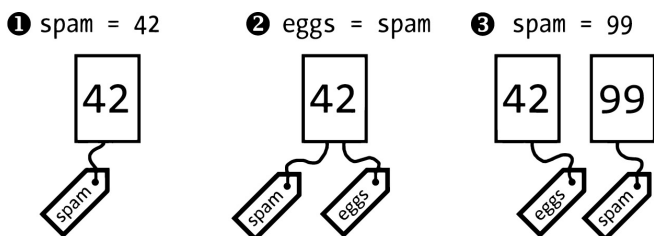


Figura 6.2. La asignación de una variable no reescribe el valor, cambia la referencia.

El cambio no afecta a `eggs`, que sigue haciendo referencia al valor 42.

Pero las listas no funcionan así, ya que los valores de lista pueden cambiar; es decir, las listas son mutables. El siguiente código facilitará la comprensión de esta distinción. Escríbelo en el intérprete de comandos interactivo:

```
1 >>> spam = [0, 1, 2, 3]
2 >>> eggs = spam # Se copia la referencia, que no la lista.
3 >>> eggs[1] = 'Hello!' # Esto cambia el valor de la lista.
>>> spam
[0, 'Hello!', 2, 3]
>>> eggs # La variable eggs se refiere a la misma lista.
[0, 'Hello!', 2, 3]
```

Este código puede parecerle extraño. Solo toca la lista `eggs`, pero parece que han cambiado tanto `eggs` como `spam`.

Cuando creamos la lista 1, le asignamos una referencia en la variable `spam`. Pero la siguiente línea copia solo la referencia de `spam` a `eggs` 2, no el valor de la lista en sí. Sigue habiendo solo una lista y ahora tanto `spam` como `eggs` se refieren a ella. La razón por la que solo hay una lista subyacente es que la lista en sí no se ha copiado. Por eso, al modificar el primer elemento de `eggs` 3,

estamos modificando la misma lista a la que hace referencia spam. Lo vemos en la figura 6.3.

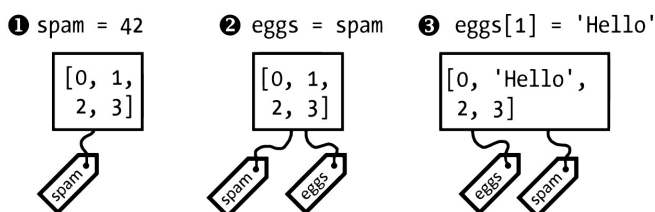


Figura 6.3. Como `spam` y `eggs` se refieren a la misma lista, un cambio en una modifica la otra.

La cosa se complica un poco porque las listas no contienen una secuencia de valores directamente, sino una secuencia de referencias a valores. Explicaré más esto en la siguiente página.

Aunque, técnicamente, las variables de Python contienen referencias a valores, la gente suele decir que la variable contiene el valor, pero ten en cuenta estas dos reglas:

- En Python, las variables nunca contienen valores, solo contienen referencias a valores.
- En Python, el operador de asignación `=` copia solo referencias, nunca copia valores.

En líneas generales, no necesitas conocer estos detalles, pero, a veces, estas sencillas reglas tienen efectos sorprendentes y deberías entender exactamente qué está haciendo Python.

Argumentos

Las referencias son especialmente importantes para entender cómo pasan los argumentos a las funciones. Cuando se llama a una función, Python copia las variables del parámetro en los argumentos. Para valores mutables como las listas (y los diccionarios, que veremos en el capítulo 7), esto significa que el código de la función modifica el valor original *in situ*. Para ver las consecuencias de esto, abre una ventana nueva en el editor de archivos, escribe el siguiente código y guarda como

passingReference.py:

```
def eggs(some_parameter):
```

```
    some_parameter.append('Hello')
```

```
spam = [1, 2, 3]
```

```
eggs(spam)
```

```
print(spam) # Imprime [1, 2, 3, 'Hello']
```

Observa que al llamar a `eggs()`, un valor de retorno no asigna un nuevo valor a `spam`. En su lugar, modifica directamente la lista *in situ*. Al ejecutar este programa, la salida es `[1, 2, 3, 'Hello']`.

Incluso aunque `spam` y `some_parameter` contengan referencias separadas, ambas se refieren a la misma lista. Por eso, la llamada al método `append('Hello')` dentro de la función afecta a la lista incluso después de que haya vuelto la llamada.

Recuerda este comportamiento. Olvidar que Python maneja así las variables de listas y diccionarios puede conducir a un comportamiento inesperado y fallos confusos.

Las funciones `copy()` y `deepcopy()`

Aunque pasar referencias suele ser la forma más útil de tratar con listas y diccionarios, si la función modifica la lista o el diccionario que se le pasa, es posible que no queramos esos cambios en el valor de lista o diccionario original. Para controlar este comportamiento, Python ofrece un módulo llamado `copy` que ofrece las funciones `copy()` y `deepcopy()`. La primera, `copy.copy()`, puede hacer un duplicado de un valor mutable como una lista o un diccionario, no solo una copia de la referencia. Escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > import copy
```

```
> > > spam = ['A', 'B', 'C']
```

```
> > > cheese = copy.copy(spam) # Crea un duplicado de la
```

lista.

```
> > > cheese[1] = 42 # Cambia cheese.
```

```
> > > spam # La variable spam no cambia.
```

```
['A', 'B', 'C']
```

```
> > > cheese # La variable cheese cambia.
```

```
['A', 42, 'C']
```

Ahora las variables `spam` y `cheese` hacen referencia a listas separadas, que es por lo que solo se modifica la lista de `cheese` cuando asignamos 42 al índice 1. Igual que las variables hacen referencia a valores en lugar de contener valores, las listas contienen referencias a valores en vez de los valores en sí. La figura 6.4 ilustra esto.

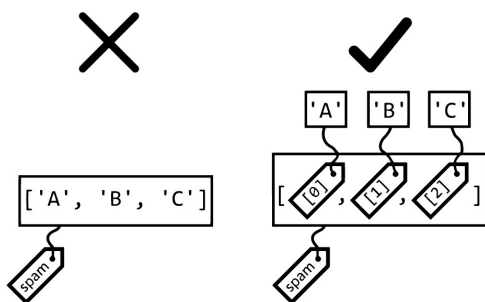


Figura 6.4. Las listas no contienen valores directamente (izquierda), sino referencias a valores (derecha).

Si la lista que necesitas copiar contiene listas, utiliza la función `copy.deepcopy()` en vez de `copy.copy()`. La función `copy.deepcopy()` copiará también estas listas internas.

Un programa corto: el salvapantallas de *Matrix*

En la película de *hackers* de ciencia ficción *Matrix*, los monitores de los ordenadores muestran un torrente de números verdes brillantes, como una lluvia digital contra una ventana de cristal. Los números pueden ser insignificantes, pero queda guay. Solo por diversión,

vamos a crear nuestro propio salvapantallas de *Matrix* en Python. Escribe el siguiente código en un archivo nuevo y guárdalo como `matrixscreensaver.py`:

```
import random, sys, time

WIDTH = 70 # El número de columnas.

try:

    # Para cada columna, cuando el contador es 0, no se muestra
    torrente.

    # De lo contrario, actúa como contador de las veces que debería
    mostrarse

    # un 1 o un 0 en esa columna.

    columns = [0] * WIDTH

    while True:

        # Bucle por cada columna:

        for i in range(WIDTH):

            if random.random() < 0.02:

                # Reinicia un contador del torrente en esta columna.

                # La longitud del torrente es de entre 4 y 14 caracteres.

                columns[i] = random.randint(4, 14)

                # Imprime un carácter en esta columna:

                if columns[i] == 0:

                    # Cambia este ‘ ‘ por ‘.’ para ver los espacios en blanco:

                    print(‘ ‘, end=”)

                else:
```


Imprime 0 o 1:

```
print(random.choice([0, 1]), end = "")
```

columns[i] -= 1 # Disminuye el contador para esta columna.

print() # Imprime una línea nueva al final de la fila de columnas.

time.sleep(0.1) # Cada fila hace una pausa de una décima de segundo.

except KeyboardInterrupt:

sys.exit() # Cuando se pulse Control-C, finaliza el programa.

Cuando ejecutes este programa, produce un torrente de unos y ceros binarios, como en la figura 6.5.

Al igual que los programas anteriores de espiga y zigzag, este crea una animación imprimiendo filas de texto en un bucle infinito que se detiene cuando el usuario pulsa **Control-C**. La principal estructura de datos de este programa es la lista `columns`, que contiene 70 enteros, uno por cada columna de salida. Cuando un entero de `columns` es 0, imprime un espacio en blanco para esa columna. Cuando es mayor que 0, imprime aleatoriamente 0 o 1 y luego disminuye el entero. Una vez que este llega a 0, esa columna vuelve a imprimir un espacio en blanco. El programa establece aleatoriamente los enteros de `columns` como enteros entre 4 y 14 para producir un torrente de unos y ceros binarios.

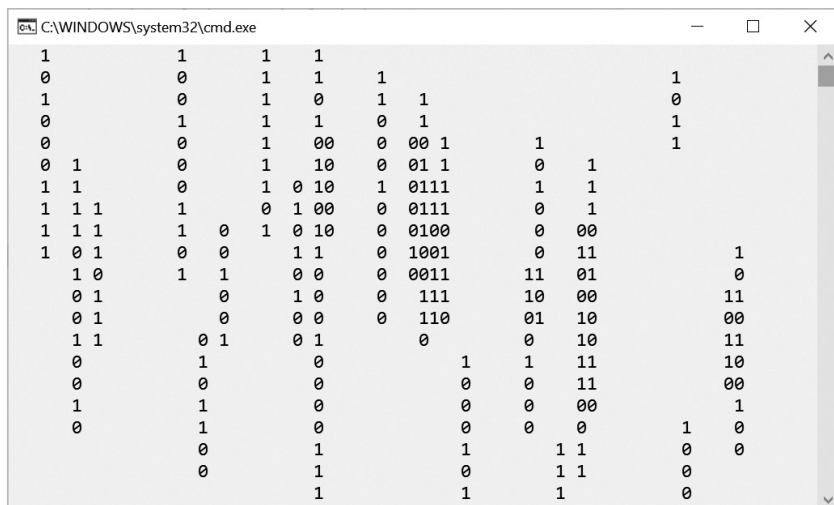
A screenshot of a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". The window displays a Matrix-style pattern of 0s and 1s. The pattern consists of multiple rows of characters, where each character is either a '0' or a '1'. The overall shape of the pattern is roughly rectangular, with some characters missing, giving it a sparse, digital appearance. The pattern is centered in the window.

Figura 6.5. El programa salvapantallas de Matrix.

Echemos un vistazo a cada parte del programa:

```
import random, sys, time
```

```
WIDTH = 70 # El número de columnas.
```

Importamos el módulo `random` para las funciones `choice()` y `randint()`, el módulo `sys` para la función `exit()` y el módulo `time` para la función `sleep()`. También configuramos una variable llamada `WIDTH` a 70 para que el programa produzca una salida de 70 columnas de caracteres. Eres libre de cambiar este valor por un entero mayor o menor en función del tamaño de la ventana en la que vas a ejecutar el programa.

La variable `WIDTH` tiene el nombre en mayúsculas porque es una variable constante. Una constante es una variable que se supone que no va a cambiar el código una vez configurada. Utilizar constantes permite escribir código más legible, como `columns = [0 * WIDTH` en vez de `columns = [0 * 70`, que puede dejarte preguntándote qué es ese 70 cuando releas el código después. En Python, nada impide que cambies el valor de una constante, pero el nombre en mayúsculas recuerda al programador que no lo haga.

El grueso del programa va dentro de un bloque `try`, que actúa si el

usuario pulsa **Control-C** para lanzar una excepción
KeyboardInterrupt:

try:

Para cada columna, cuando el contador es 0, no se muestra
torrente.

De lo contrario, actúa como contador de las veces que debería
mostrarse

un 1 o un 0 en esa columna.

columns = [0] * WIDTH

La variable columns contiene una lista de 0 enteros. El número de enteros de esta lista es igual a WIDTH. Cada uno de estos enteros controla si una columna de la ventana de salida imprime un torrente de números binarios o no:

while True:

Bucle por cada columna:

for i in range(WIDTH):

if random.random() < 0.02:

Reinicia un contador del torrente en esta columna.

La longitud del torrente es de entre 4 y 14 caracteres.

columns[i] = random.randint(4, 14)

Queremos que este programa se ejecute para siempre, así que lo ponemos todo en un bucle while True: infinito. Dentro de este bucle, hay un bucle for que itera por cada columna de una sola fila. La variable de bucle i representa los índices de columnas; empieza en 0 y va subiendo, pero no incluye WIDTH. El valor de columns[0] representa lo que debería imprimirse en la columna que queda más a la izquierda, columns[1] hace lo propio para la siguiente columna por la izquierda y así sucesivamente.

Para cada columna, hay una probabilidad del dos por ciento de que el entero de `columns[i]` se establezca en un número entre 4 y 14. Calculamos esta probabilidad comparando `random.random()` (una función que devuelve un flotante aleatorio entre 0.0 y 1.0) con 0.02. Si quieres que el torrente sea más o menos denso, puedes aumentar o reducir este número, respectivamente. Establecemos el contador para cada columna en un número aleatorio entre 4 y 14:

```
# Imprime un carácter en esta columna:
```

```
if columns[i] == 0:
```

```
# Cambia este ‘ ‘ por ‘.’ para ver los espacios en blanco:
```

```
print(‘ ‘, end=’ ’)
```

```
else:
```

```
# Imprime 0 o 1:
```

```
print(random.choice([0, 1]), end=’ ’)
```

```
columns[i] -= 1 # Disminuye el contador para esta columna.
```

También dentro del bucle `for`, el programa determina si debería imprimir un número binario 0 o 1 aleatorio o un espacio en blanco. Si `columns[i]` es 0, imprime un espacio en blanco. De lo contrario, pasa `list [0, 1]` a la función `random.choice()`, que devuelve un valor aleatorio de esa lista para imprimir. El código también reduce el contador de `columns[i]` para que se acerque a 0 y dejen de imprimirse números binarios.

Si quieres ver los espacios “en blanco” que imprime el programa, prueba a cambiar la cadena ‘ ‘ por ‘.’ y vuelve a ejecutar el programa. La salida será parecida a esto:

```
.....1.....  
.....0.....1.....1.....  
.....1.....0.....1.....0.....
```

.....1...0.....0.....0.....1....0.....

.....1.1.1.....0.....0.....1....1..1.....

.....0.0.0.....0.....1.....00....1..1.....

Cuando termina el bloque else, también termina el bloque for:

print() # Imprime una línea nueva al final de la fila de columnas.

time.sleep(0.1) # Cada fila hace una pausa de una décima de segundo.

except KeyboardInterrupt:

sys.exit() # Cuando se pulse Control-C, finaliza el programa.

La llamada a print() después del bucle for imprime una línea nueva, ya que las llamadas anteriores a print() para cada columna pasan el argumento end = " para evitar que se impriman líneas nuevas después de cada columna. Por cada fila impresa, el programa introduce una pausa de una décima de segundo llamando a time.sleep(0.1).

La última parte del programa es un bloque except que sale del programa si el usuario pulsa **Control-C** para lanzar una excepción KeyboardInterrupt.

Resumen

Las listas son tipos de datos útiles, ya que permiten escribir código que funciona con un número modificable de valores en una única variable. Más adelante en el libro, veremos programas que usan listas para hacer cosas que de otro modo serían difíciles o imposibles.

Una lista es un tipo de datos de secuencia que es mutable, lo que significa que puede cambiar. Las tuplas y las cadenas también son tipos de datos de secuencia, pero son inmutables y no pueden cambiar. Podemos sobrescribir una variable que contiene una tupla o una cadena con un nuevo valor de tupla o cadena, que no es lo

mismo que modificar el valor existente *in situ*, como hacen los métodos `append()` o `remove()` con las listas.

Las variables no almacenan valores de lista directamente, sino que guardan referencias a listas. Es una distinción importante cuando se copian variables o se pasan listas como argumentos en llamadas a funciones. Como el valor que se está copiando es la referencia a la lista, cualquier cambio que se haga en la lista puede afectar a otra variable del programa. Podemos usar `copy()` o `deepcopy()` si queremos hacer cambios en una lista en una variable sin modificar la lista original.

Preguntas prácticas

1. ¿Qué es `[]`?
2. ¿Cómo asignarías el valor `'hello'` como el tercer valor de una lista guardada en una variable llamada `spam`? (Supongamos que `spam` contiene `[2, 4, 6, 8, 10]`).

Para las tres preguntas siguientes, imagina que `spam` contiene la lista `['a', 'b', 'c', 'd']`.

3. ¿A qué se evalúa `spam[int(int('3' * 2) // 11)]`?
4. ¿A qué se evalúa `spam[-1]`?
5. ¿A qué se evalúa `spam[:2]`?

Para las tres preguntas siguientes, imagina que `bacon` contiene la lista `[3.14, 'cat', 11, 'cat', True]`.

6. ¿A qué se evalúa `bacon.index('cat')`?
7. ¿Cómo hace `bacon.append(99)` que quede el valor de lista de `bacon`?
8. ¿Cómo hace `bacon.remove('cat')` que quede el valor de lista de `bacon`?
9. ¿Cuáles son los operadores para concatenar y replicar listas?

- 10.¿Cuál es la diferencia entre los métodos de lista `append()` e `insert()`?
- 11.¿Qué dos formas hay de eliminar valores de una lista?
- 12.Nombra algunas similitudes entre los valores de lista y los valores de cadena.
- 13.¿Cuál es la diferencia entre listas y tuplas?
- 14.¿Cómo escribes el valor de tupla que tiene dentro solo el valor entero 42?
- 15.¿Cómo se puede poner un valor de lista en forma de tupla? ¿Y un valor de tupla en forma de lista?
- 16.Las variables que “contienen” valores de lista no contienen en realidad listas. ¿Qué contienen?
- 17.¿Cuál es la diferencia entre `copy.copy()` y `copy.deepcopy()`?

Programas de práctica

Para practicar, escribe programas que realicen las siguientes tareas.

Poner comas

Supongamos que tienes un valor de lista como este:

```
spam = ['apples', 'bananas', 'tofu', 'cats']
```

Escribe una función que coja un valor de lista como argumento y devuelva una cadena con todos los elementos separados por una coma y un espacio, con “and” insertado antes del último elemento. Por ejemplo, pasar la lista `spam` anterior a la función devolvería ‘apples, bananas, tofu, and cats’. Pero tu función debería ser capaz de trabajar con cualquier valor de lista que se le pase. Asegúrate de probar el caso cuando se pase una lista vacía [a tu función.

Lanzamientos de moneda

Para este ejercicio, vamos a hacer un experimento. Si lanzas una moneda al aire 100 veces y anotas una H (*head*) cada vez que salga cara y una T (*tail*) cada vez que salga cruz, crearás una lista parecida a T T T T H H H H T T. Si pides a un humano que invente el resultado aleatorio de cien lanzamientos de moneda, es probable que termines con un resultado alterno de caras y cruces como H T H T H H T H T T, que es aleatorio a ojos humanos, pero no lo es matemáticamente. Un humano casi nunca escribirá una racha de seis caras o seis cruces seguidas, aunque es muy probable que salga eso si tiras una moneda de verdad. A los humanos se nos da previsiblemente mal ser aleatorios.

Escribe un programa que descubra con qué frecuencia sale una racha de seis caras o seis cruces en una lista de 100 caras y cruces generada aleatoriamente. El programa debería descomponer el experimento en dos partes: la primera genera una lista de 100 valores 'H' y 'T' seleccionados aleatoriamente y la segunda comprueba si hay una racha en ella. Pon todo este código en un bucle que repita el experimento 10 000 veces para ver qué porcentaje de lanzamientos contiene una racha de seis caras o seis cruces seguidas. Una pista: la llamada a la función `random.randint(0, 1)` devolverá un valor 0 la mitad de las veces y un valor 1 la otra mitad.

Puedes empezar con esta plantilla:

```
import random

number_of_streaks = 0

for experiment_number in range(10000): # Ejecuta 10 000
    experimentos en total

    # Código que crea una lista de 100 valores 'heads' o 'tails'.

    # Código que comprueba si hay una racha de 6 caras o 6 cruces
    seguidas.

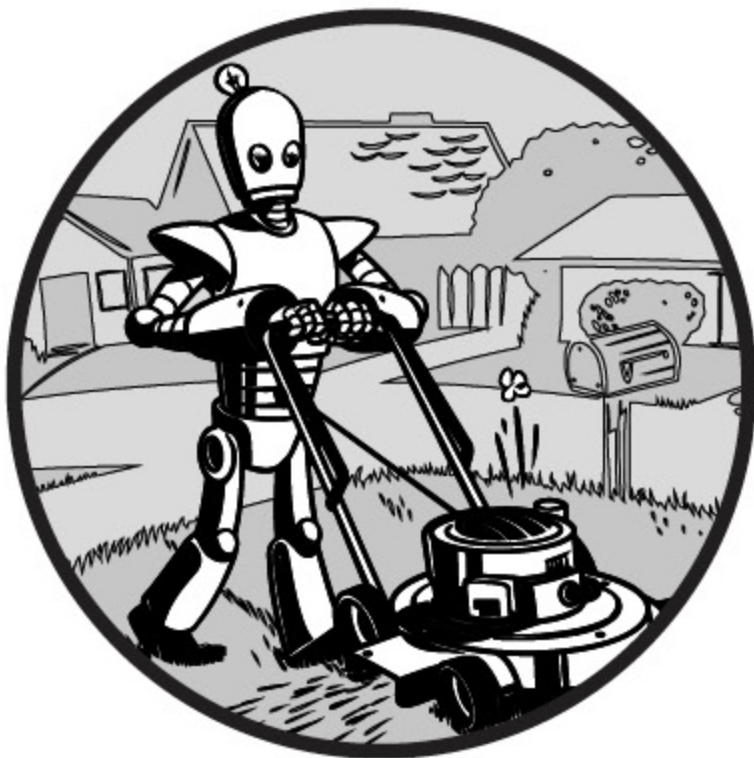
    print('Chance of streak: %s%%' % (number_of_streaks / 100))
```

Naturalmente, esto es solo un cálculo, pero 10.000 es un tamaño de

muestra decente. Algunos conocimientos matemáticos te darían la respuesta exacta y te ahorrarían el trabajo de escribir un programa, pero a los programadores se nos dan bastante mal las matemáticas.

Para crear la lista, usa un bucle for que añada a la lista 'H' o 'T' aleatoriamente 100 veces. Para determinar si hay una racha de seis caras o seis cruces, crea una rebanada como `some_list[i:i + 6]` (que contiene los seis elementos a partir del índice `i`) y compárala con los valores de lista `['H', 'H', 'H', 'H', 'H', 'H']` y `['T', 'T', 'T', 'T', 'T', 'T']`.

Diccionarios y estructuración de datos



Este capítulo aborda el tipo de datos diccionario, que ofrece una manera flexible de acceder a datos y organizarlos. Combinando los diccionarios con el conocimiento de las listas que has adquirido en el capítulo anterior, también aprenderás a crear una estructura de datos para modelar un tablero de ajedrez.

El tipo de datos diccionario

Al igual que una lista, un diccionario es una colección mutable de varios valores. Pero, a diferencia de los índices de las listas, los índices de los diccionarios pueden usar muchos tipos de datos, no

solo enteros. Estos índices de diccionario se llaman claves y una clave con su valor asociado se conoce como par clave-valor.

En código, un diccionario va entre llaves ({}). Escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > my_cat = {'size': 'fat', 'color': 'gray', 'age': 17}
```

Esto asigna un diccionario a la variable `my_cat`. Las claves de este diccionario son `'size'`, `'color'` y `'age'`. Los valores para estas claves son `'fat'`, `'gray'` y `17`, respectivamente. Podemos acceder a los valores a través de sus claves:

```
> > > my_cat['size']
```

```
'fat'
```

```
> > > 'My cat has ' + my_cat['color'] + ' fur.'
```

```
'My cat has gray fur.'
```

Usando diccionarios, podemos almacenar varios datos sobre lo mismo en una única variable. Esta variable `my_cat` contiene tres cadenas diferentes que describen a mi gato y puedo usarla como argumento o valor de retorno en una llamada a función, evitándome tener que crear tres variables diferentes.

Los diccionarios también pueden usar valores enteros como clave, igual que las listas usan enteros como índices, pero no tienen por qué empezar en 0 y pueden ser cualquier número:

```
> > > spam = {12345: 'Luggage Combination', 42: 'The Answer'}
```

```
> > > spam[12345]
```

```
'Luggage Combination'
```

```
> > > spam[42]
```

```
'The Answer'
```

```
> > > spam[0]
```

Traceback (most recent call last):

File "<python-input-0>", line 1, in <module>

KeyError: 0

Los diccionarios tienen claves, no índices. En este ejemplo, aunque el diccionario tiene claves que son enteros (12345 y 42), no tiene un índice de 0 a 41 como tendría una lista.

Comparación de listas y diccionarios

A diferencia de las listas, los elementos de los diccionarios están desordenados. El primer elemento de una lista llamada spam sería spam[0]. Pero no hay "primer" elemento en un diccionario. Aunque el orden de los elementos es importante para determinar si dos listas son la misma, podemos introducir pares clave-valor en un diccionario en cualquier orden. Introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > spam = ['cats', 'dogs', 'moose']
```

```
> > > bacon = ['dogs', 'moose', 'cats']
```

```
> > > spam = bacon # El orden de los elementos de la lista importa.
```

False

```
> > > eggs = {'name': 'Zophie', 'species': 'cat', 'age': '8'}
```

```
> > > ham = {'species': 'cat', 'age': '8', 'name': 'Zophie'}
```

```
> > > eggs = ham # El orden de los pares clave-valor del diccionario no importa.
```

True

Los diccionarios eggs y ham son valores idénticos, aunque hayamos introducido sus pares clave-valor en distinto orden. Como los

diccionarios no están ordenados, no son tipos de datos de secuencia y no se puede hacer rebanadas como en una lista.

Intentar acceder a una clave que no existe en un diccionario producirá un mensaje de error `KeyError`, del estilo del mensaje de error `IndexError` de las listas. Escribe lo siguiente en el intérprete de comandos interactivo y observa el error que aparece porque falta la clave `'color'`:

```
> > > spam = {'name': 'Zophie', 'age': 7}
```

```
> > > spam['color']
```

Traceback (most recent call last):

File "<python-input-0>", line 1, in <module>

```
spam['color']
```

`KeyError: 'color'`

Aunque los diccionarios no estén ordenados, el hecho de poder usar valores arbitrarios como claves nos permite organizar los datos de maneras potentes. Supongamos que queremos que nuestro programa guarde datos sobre los cumpleaños de nuestros amigos. Podemos usar un diccionario con sus nombres como claves y los cumpleaños como valores. Abre una ventana nueva en el editor de archivos, escribe el siguiente código y guárdalo como `birthdays.py`:

```
1 birthdays = {'Alice': 'Apr 1', 'Bob': 'Dec 12', 'Carol': 'Mar 4'}
```

```
while True:
```

```
    print('Enter a name: (blank to quit)')
```

```
    name = input()
```

```
    if name == '':
```

```
        break
```

```
    2 if name in birthdays:
```

```
3 print(birthdays[name] + ' is the birthday of ' + name)
```

```
else:
```

```
print('I do not have birthday information for ' + name)
```

```
print('What is their birthday?')
```

```
bday = input()
```

```
4 birthdays[name] = bday
```

```
print('Birthday database updated.')
```

El código crea un diccionario inicial y guarda en él birthdays 1. Puedes ver si el nombre introducido existe como clave en el diccionario con la palabra clave in 2, igual que con las listas. Si el nombre está en el diccionario, accedes al valor asociado usando corchetes 3; si no, puedes añadirlo usando la misma sintaxis de corchetes combinada con el operador de asignación 4. Al ejecutar el programa, saldrá esto:

Enter a name: (blank to quit)

Alice

Apr 1 is the birthday of Alice

Enter a name: (blank to quit)

Eve

I do not have birthday information for Eve

What is their birthday?

Dec 5

Birthday database updated.

Enter a name: (blank to quit)

Eve

Dec 5 is the birthday of Eve

Enter a name: (blank to quit)

Naturalmente, todos los datos que introduzcas en este programa se olvidan cuando termine el programa. Aprenderás a guardar datos en archivos del disco duro en el capítulo 10.

Devolución de claves y valores

Hay tres métodos de diccionario que devuelven una lista de valores de las claves del diccionario, los valores o tanto claves como valores: `keys()`, `values()` e `items()`. Los valores devueltos por estos métodos no son listas realmente: no se pueden modificar y no disponen de un método `append()`. Pero estos tipos de datos (`dict_keys`, `dict_values` y `dict_items`, respectivamente) pueden usarse en bucles `for`. Para ver cómo funcionan estos métodos, pon esto en el intérprete de comandos interactivo:

```
> > > spam = {'color': 'red', 'age': 42}
```

```
> > > for v in spam.values():
```

```
... print(v)
```

```
red
```

```
42
```

Aquí, un bucle `for` itera por cada uno de los valores del diccionario `spam`. Un bucle `for` también puede iterar por las claves o por las claves y los valores:

```
> > > for k in spam.keys():
```

```
... print(k)
```

```
color
```

```
age
```

```
> > > 'color' in spam.keys()
```

True

```
> > > 'age' not in spam.keys()
```

False

```
> > > 'red' in spam.values()
```

True

```
> > > for i in spam.items():
```

```
... print(i)
```

```
('color', 'red')
```

```
('age', 42)
```

Cuando usamos los métodos `keys()`, `values()` e `items()`, un bucle `for` puede iterar por las claves, los valores o los pares clave-valor del diccionario, respectivamente, y podemos usar los operadores `in` y `not in` para determinar si un valor existe como clave o valor en el diccionario. Observa que los valores del valor `dict_items` devuelto por el método `items()` son tuplas de la clave y el valor.

También podemos usar los operadores `in` y `not in` con el valor del propio diccionario para comprobar la existencia de una clave. Es el equivalente a usar estos operadores con el método `keys()`:

```
> > > 'color' in spam
```

True

```
> > > 'color' in spam.keys()
```

True

Si quieres obtener una lista real a partir de uno de estos métodos, pasa su valor de retorno de estilo lista a la función `list()`. Escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > spam = {'color': 'red', 'age': 42}
```



```
> > > spam.keys() # Devuelve un valor dict_keys de estilo lista.
```

```
dict_keys(['color', 'age'])
```

```
> > > list(spam.keys()) # Devuelve un valor de lista real.
```

```
['color', 'age']
```

La línea `list(spam.keys())` coge el valor `dict_keys` devuelto por `keys()` y se lo pasa a `list()`, que devuelve el valor de lista `['color', 'age']`.

También se puede usar el truco de la asignación múltiple en un bucle `for` para asignar la clave y el valor a variables separadas. Escribe esto en el intérprete de comandos interactivo:

```
> > > spam = {'color': 'red', 'age': 42}
```

```
> > > for k, v in spam.items():
```

```
... print('Key: ' + str(k) + ' Value: ' + str(v))
```

```
Key: color Value: red
```

```
Key: age Value: 42
```

Este código crea un diccionario con las claves `'color'` y `'age'` cuyos valores son `'red'` y `42`, respectivamente. El bucle `for` itera por las tuplas devueltas por el método `items()`: `('color', 'red')` y `('age', 42)`. A las dos variables, `k` y `v`, se les asigna el primer valor (la clave) y el segundo valor (el valor) de estas tuplas. El cuerpo del bucle imprime las variables `k` y `v` para cada par clave-valor.

Aunque podemos usar muchos valores para estas claves, no se puede usar una lista o diccionario como clave de un diccionario. Estos tipos de datos no son aptos para *hashing*, un concepto que queda fuera del ámbito de este libro. Si necesitas una lista como clave de diccionario, usa una tupla en su lugar.

Comprobar si existe una clave

Comprobar si existe una clave en un diccionario antes de acceder al valor de esa clave puede resultar tedioso. Por suerte, los

diccionarios cuentan con un método `get()` que toma dos argumentos: la clave del valor que recuperar y un valor de respaldo que devolver si no existe esa clave.

Escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > picnic_items = {'apples': 5, 'cups': 2}
```

```
> > > 'I am bringing ' + str(picnic_items.get('cups', 0)) + ' cups.'
```

```
'I am bringing 2 cups.'
```

```
> > > 'I am bringing ' + str(picnic_items.get('eggs', 0)) + ' eggs.'
```

```
'I am bringing 0 eggs.'
```

Como no hay clave 'eggs' en el diccionario `picnic_items`, el método `get()` devuelve el valor predeterminado 0. Sin usar `get()`, el código habría provocado un mensaje de error, como en el siguiente ejemplo:

```
> > > picnic_items = {'apples': 5, 'cups': 2}
```

```
> > > 'I am bringing ' + str(picnic_items['eggs']) + ' eggs.'
```

Traceback (most recent call last):

File "<python-input-0>", line 1, in <module>

```
'I am bringing ' + str(picnic_items['eggs']) + ' eggs.'
```

```
KeyError: 'eggs'
```

Comprobar la existencia de una clave antes de acceder a su valor puede evitar que tus programas fallen con un mensaje de error.

Establecer valores predeterminados

Con frecuencia, hay que establecer un valor predeterminado para una clave solo si esa clave no tiene ya un valor. Supongamos que

tenemos este código:

```
> > > spam = {'name': 'Pooka', 'age': 5}
```

```
> > > if 'color' not in spam:
```

```
... spam['color'] = 'black'
```

```
.
```

```
> > > spam
```

```
{'name': 'Pooka', 'age': 5, 'color': 'black'}
```

El método `setdefault()` ofrece una manera de hacer eso con una línea de código. El primer argumento que se pasa al método es la clave que hay que comprobar y el segundo, el valor para establecer si esa clave no existe. Si existe, el método `setdefault()` devuelve el valor de la clave. Escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > spam = {'name': 'Pooka', 'age': 5}
```

```
> > > spam.setdefault('color', 'black') # Establece la clave  
'color' en 'black'.
```

```
'black'
```

```
> > > spam
```

```
{'name': 'Pooka', 'age': 5, 'color': 'black'}
```

```
> > > spam.setdefault('color', 'white') # No hace nada.
```

```
'black'
```

```
> > > spam
```

```
{'name': 'Pooka', 'age': 5, 'color': 'black'}
```

La primera vez que llamamos a `setdefault()`, el diccionario de `spam` cambia a `{'color': 'black', 'age': 5, 'name': 'Pooka'}`. El método devuelve el valor `'black'` porque ese es ahora el valor establecido

para la clave 'color'. Cuando se llama a `spam.setdefault('color', 'white')` a continuación, el valor no cambia a 'white', porque `spam` ya tiene una clave llamada 'color'.

El método `setdefault()` es un buen atajo para asegurarse de que existe una clave. Aquí tenemos un programa corto que cuenta el número de veces que aparece cada letra en una cadena. Abre la ventana del editor de archivos e introduce el siguiente código, guardándolo como `characterCount.py`:

```
message = 'It was a bright cold day in April, and the clocks were  
striking thirteen.'
```

```
count = {}
```

```
for character in message:
```

```
    count.setdefault(character, 0) 1
```

```
    count[character] = count[character] + 1 2
```

```
print(count)
```

El programa pasa en bucle por cada carácter de la cadena de la variable `message`, contando con qué frecuencia aparece cada uno. La llamada al método `setdefault()` 1 se asegura de que la clave está en el diccionario `count` (con un valor predeterminado de 0) para que el programa no dé un error `KeyError` cuando se ejecute `count[character] = count[character] + 1` 2. Al ejecutar el programa, la salida quedará así:

```
{'I': 1, 't': 6, ' ': 13, 'w': 2, 'a': 4, 's': 3, 'b': 1, 'r': 5, 'i': 6,
```

```
'g': 2, 'h': 3, 'c': 3, 'o': 2, 'l': 3, 'd': 3, 'y': 1, 'n': 4, 'A': 1,
```

```
'p': 1, ',': 1, 'e': 5, 'k': 2, '.': 1}
```

Por la salida, vemos que la `c` minúscula aparece tres veces, el carácter de espacio trece y la `A` mayúscula, solo una vez. Este programa funcionará al margen del mensaje que haya en la variable `message`, ¡incluso aunque la cadena tenga millones de caracteres!

Modelar cosas del mundo real con estructuras de datos

Incluso antes de Internet, era posible jugar al ajedrez con alguien al otro lado del mundo. Cada jugador colocaba un tablero en su casa y, por turnos, se enviaban postales describiendo sus movimientos. Para hacer eso, los jugadores necesitaban una manera que describiese sin lugar a ambigüedades el estado del tablero y sus movimientos.

En la notación algebraica del ajedrez, los cuadros del tablero se identifican con una coordenada de número y letra, como en la figura 7.1.

	a	b	c	d	e	f	g	h	
8	a8	b8	c8	d8	e8	f8		h8	8
7		b7	c7	d7	e7	f7	g7		7
6	a6	b6	c6	d6	e6		g6	h6	6
5	a5	b5	c5	d5	e5	f5	g5	h5	5
4	a4	b4	c4	d4	e4	f4	g4	h4	4
3	a3	b3	c3	d3	e3		g3	h3	3
2	a2	b2	c2	d2	e2	f2	g2		2
1	a1	b1		d1	e1	f1	g1	h1	1
	a	b	c	d	e	f	g	h	

Figura 7.1. Las coordenadas de un tablero de ajedrez en notación algebraica.

Para las piezas se utilizan letras en función del idioma, por ejemplo: R o K para el rey (*king*), D o Q para la dama/reina (*queen*), T o R para la torre (*rook*), A o B para el alfil (*bishop*) y C o N para el caballo (*knight*). Describir un movimiento requiere especificar la letra de la pieza y la coordenada de destino. Un par de estos movimientos describe lo que pasa en una sola jugada (yendo primero las blancas); por ejemplo, la notación “2. Nf3 Nc6” indica que el caballo blanco se mueve a f3 y el caballo negro a c6 en el siguiente turno.

La notación algebraica es un poco más compleja que esto, pero

ahora mismo podemos describir sin ambigüedad un juego de ajedrez sin necesidad de estar delante de un tablero. ¡Nuestro contrincante puede incluso estar al otro lado del mundo! De hecho, no hace falta ni siquiera instalar un tablero físico si tienes buena memoria: puedes leer los movimientos enviados por correo y actualizar el tablero con tu imaginación.

Los ordenadores tienen buena memoria. Un programa en un ordenador moderno puede almacenar fácilmente miles de millones de cadenas como '2. Nf3 Nc6'. Así es como los ordenadores pueden jugar al ajedrez sin contar con un tablero físico. Modelan datos para representar un tablero y nosotros podemos escribir código que funcione con ese modelo para simular una partida de ajedrez.

Aquí es donde las listas y los diccionarios nos vienen bien. Por ejemplo, podríamos usar nuestra propia notación para que el diccionario de Python {'h1': 'bK', 'c6': 'wQ', 'g2': 'bB', 'h5': 'bQ', 'e3': 'wK'} represente el tablero de ajedrez de la figura 7.2.

Vamos a usar este esquema de estructura de datos para crear nuestro propio programa de tablero de ajedrez interactivo.

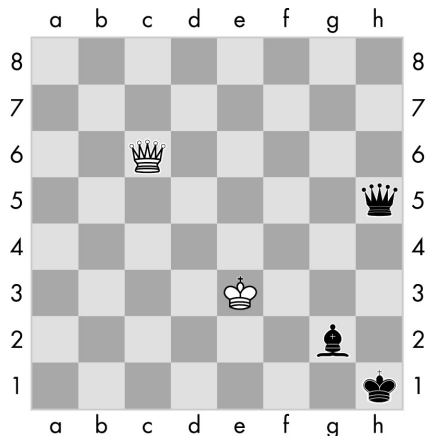


Figura 7.2. Un tablero de ajedrez modelado por el diccionario {'h1': 'bK', 'c6': 'wQ', 'g2': 'bB', 'h5': 'bQ', 'e3': 'wK'}.

Proyecto 1: Simulador de un tablero de ajedrez interactivo

Incluso los primeros ordenadores hacían cálculos mucho más deprisa que cualquier humano, pero, por aquel entonces, la gente consideraba el ajedrez como la verdadera demostración de la inteligencia computacional.

No vamos a crear nuestro propio programa de ajedrez aquí (¡porque eso requeriría un libro entero!), pero podemos crear un programa de tablero interactivo con lo que hemos tratado hasta ahora.

No hace falta conocer las reglas del ajedrez para este programa. Basta con saber que se juega en un tablero de 8×8 con fichas negras y blancas llamadas peones, caballeros, alfiles, torres, reinas y reyes. El cuadro superior izquierdo y el inferior derecho del tablero deberían ser blancos y nuestro programa asume que el fondo de la ventana de salida es negro (y no blanco como sería si usásemos un papel). Nuestro programa de tablero de ajedrez es solo un tablero con piezas encima, no vamos a imponer las reglas para el movimiento de las piezas. Utilizaremos caracteres de texto para “dibujar” un tablero de ajedrez, como el que se muestra en la figura 7.3.

Los gráficos estarían bien y harían las piezas más fácilmente identificables, pero hemos recogido toda la información sobre las piezas sin ellas. Este enfoque basado en texto nos permite escribir el programa con solo la función `print()` y no nos obliga a instalar ninguna librería como Pygame (de la que hablo en mi libro *Invent Your Own Computer Games with Python*, No Starch Press, 2016) para nuestro programa.

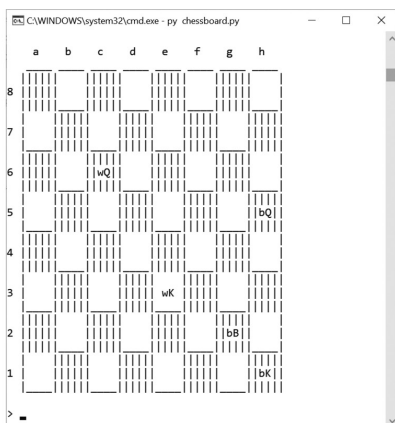
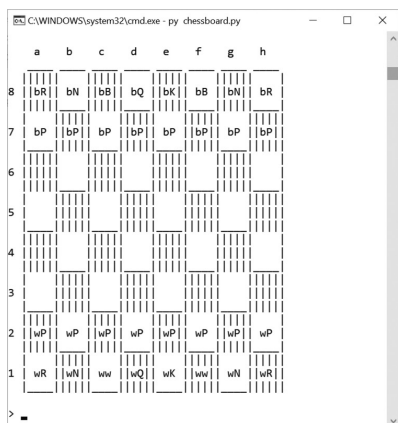


Figura 7.3. La salida no gráfica basada en texto del programa del tablero de ajedrez.

Primero, tenemos que diseñar una estructura de datos que pueda representar un tablero de ajedrez y cualquier configuración posible de piezas sobre él. El ejemplo del apartado anterior funciona: el tablero es un diccionario de Python con claves de cadena 'a1' a 'h8' para representar cuadros en el tablero. Observa que estas cadenas siempre tienen dos caracteres. Además, la letra va siempre en minúscula y antes del número. Esto es importante, usaremos esos detalles en el código.

Para representar las piezas, vamos a usar también cadenas de dos caracteres, donde la primera letra será 'w' o 'b' minúscula para indicar el color blanco (*white*) o negro (*black*) y la segunda letra será una 'P', 'N', 'B', 'R', 'Q' o 'K' mayúscula para representar el tipo de pieza. La figura 7.4 muestra cada pieza con la cadena que la representa.



Figura 7.4. Las representaciones en cadenas de dos caracteres para cada pieza de ajedrez.

Las claves del diccionario de Python identifican los cuadros del tablero y los valores la pieza en ese cuadro. La ausencia de una clave en el diccionario representa un cuadro vacío. Un diccionario va bien para almacenar esta información: las claves pueden usarse solo una vez y los cuadros del tablero solo pueden tener una pieza en cada momento.

Paso 1: configurar el programa

La primera parte del programa importa el módulo `sys` por la función

exit() y el módulo copy por la función copy(). Al empezar el juego, los jugadores blanco y negro tienen 16 piezas cada uno.

La constante STARTING_PIECES tendrá un diccionario de tablero con todas las piezas en la posición:

```
import sys, copy

STARTING_PIECES = {'a8': 'bR', 'b8': 'bN', 'c8': 'bB', 'd8': 'bQ',
                   'e8': 'bK', 'f8': 'bB', 'g8': 'bN', 'h8': 'bR', 'a7': 'bP', 'b7': 'bP',
                   'c7': 'bP', 'd7': 'bP', 'e7': 'bP', 'f7': 'bP', 'g7': 'bP', 'h7': 'bP',
                   'a1': 'wR', 'b1': 'wN', 'c1': 'wB', 'd1': 'wQ', 'e1': 'wK', 'f1': 'wB',
                   'g1': 'wN', 'h1': 'wR', 'a2': 'wP', 'b2': 'wP', 'c2': 'wP', 'd2': 'wP',
                   'e2': 'wP', 'f2': 'wP', 'g2': 'wP', 'h2': 'wP'}
```

Nota Este código es un poco difícil de escribir. Puedes copiarlo en <https://autbor.com/3/chessboard.py>. Cuando el programa necesite restablecer el tablero a la configuración inicial, puede copiar STARTING_PIECES con la función copy.copy().

Paso 2: crear una plantilla de tablero de ajedrez

La variable BOARD_TEMPLATE contendrá una cadena que actúe como plantilla para un tablero de ajedrez. El programa puede insertar las cadenas de las piezas individuales ahí antes de imprimir. Usando tres caracteres de comillas dobles seguidos, podemos crear una cadena multilínea que se extiende por varias líneas de código. La cadena multilínea termina con otros tres caracteres de comillas dobles.

Esta sintaxis de Python es más fácil que intentar encajar todo en una sola línea con caracteres de escape \n. Veremos más sobre las cadenas multilínea en el siguiente capítulo.

```
BOARD_TEMPLATE = """
```

```
a b c d e f g h
```

— — — — —

||||| ||||| ||||| ||||| |

8 ||{}| {} ||{}| {} ||{}| {} ||{}| {} |

|||||—|||||—|||||—|||||—|

| ||||| ||||| ||||| |||||

7 | {} ||{}| {} ||{}| {} ||{}| {} ||{}|

|—|||||—|||||—|||||—|||||

||||| ||||| ||||| ||||| |

6 ||{}| {} ||{}| {} ||{}| {} ||{}| {} |

|||||—|||||—|||||—|||||—|

| ||||| ||||| ||||| |||||

5 | {} ||{}| {} ||{}| {} ||{}| {} ||{}|

|—|||||—|||||—|||||—|||||

||||| ||||| ||||| ||||| |

4 ||{}| {} ||{}| {} ||{}| {} ||{}| {} |

|||||—|||||—|||||—|||||—|

| ||||| ||||| ||||| |||||

3 | {} ||{}| {} ||{}| {} ||{}| {} ||{}|

|—|||||—|||||—|||||—|||||

||||| ||||| ||||| ||||| |

2 ||{}| {} ||{}| {} ||{}| {} ||{}| {} |

|||||—|||||—|||||—|||||—|

```
| ||||| ||||| ||||| |||||
1 | {} ||{}|| {} ||{}|| {} ||{}|| {} ||{}||
|_||||_|_||||_|_||||_|_|||||
“”””
```

```
WHITE_SQUARE = '||'
```

```
BLACK_SQUARE = '‘
```

Los pares de llaves representan lugares en la cadena donde insertaremos cadenas de piezas, como ‘wR’ o ‘bQ’. Si el cuadro está vacío, el programa insertará la cadena WHITE_SQUARE o BLACK_SQUARE, que explicaré con más detalle cuando comente la función print_chessboard().

Paso 3: imprimir el tablero de ajedrez actual

Vamos a definir una función print_chessboard() que acepte el diccionario del tablero, imprima un tablero de ajedrez en la pantalla y refleje las piezas en el tablero. Llamaremos al método de cadena format() con la cadena BOARD_TEMPLATE y le pasaremos una lista de cadenas. El método format() devuelve una nueva cadena con los pares de llaves {} de BOARD_TEMPLATE sustituidos por las cadenas de la lista que se le ha pasado. Veremos más sobre format() en el próximo capítulo. Echemos un vistazo al código de print_chessboard():

```
def print_chessboard(board):
```

```
squares = []
```

```
is_white_square = True
```

```
for y in '87654321':
```

```
for x in 'abcdefgh':
```

```
#imprime (x, y, is_white_square) # DEPURA: muestra coordenadas
```

Hay 64 cuadros en un tablero de ajedrez y 64 pares de llaves en la cadena `BOARD_TEMPLATE`. Tenemos que crear una lista de 64 cadenas que sustituyan estos pares de llaves. Vamos a almacenar esa lista en la variable `squares`. Las cadenas de esta lista representan piezas de ajedrez, como `'wB'` y `'bQ'`, o cuadros vacíos. Dependiendo de si el cuadro vacío es blanco o negro, usaremos la cadena `WHITE_SQUARE` (`'|'`) o la cadena `BLACK_SQUARE` (`' '`). Utilizaremos un valor booleano en la variable `is_white_square` para llevar la cuenta de qué cuadros son blancos y cuáles son negros.

Dos bucles `for` anidados pasarán por los 64 cuadros del tablero, empezando por el superior izquierdo y yendo hacia la derecha y de arriba abajo. El cuadro de la esquina superior izquierda es blanco, así que empezaremos con `is_white_square` como `True`. Recuerda que los bucles `for` pueden pasar por los enteros dados por `range()`, el valor de una lista o los caracteres individuales de una cadena. En estos dos bucles `for`, las variables `y` y `x` toman los caracteres de las cadenas `'87654321'` y `'abcdefgh'`, respectivamente. Para ver el orden en el que el código pasa en bucle por los cuadros (junto con el color de cada cuadro), quita el comentario de la línea `print(x, y, is_white_square)` antes de ejecutar el programa.

El código dentro de los bucles `for` crea la lista `squares` con las cadenas apropiadas:

```
if x + y in board.keys():
    squares.append(board[x + y])
else:
    if is_white_square:
        squares.append(WHITE_SQUARE)
    else:
        squares.append(BLACK_SQUARE)
is_white_square = not is_white_square
is_white_square = not is_white_square
```

```
print(BOARD_TEMPLATE.format(*squares))
```

Las cadenas de las variables del bucle `x` e `y` se concatenan para formar una cadena de cuadro de dos caracteres. Por ejemplo, si `x` es `'a'` e `y` es `'8'`, `x + y` se evalúa como `'a8'`, y `x + y` in `board.keys()` comprueba si esta cadena existe como clave en el diccionario del tablero. Si existe, el código añade la cadena de la pieza para ese cuadro al final de la lista `squares`.

Si no existe, el código debe añadir la cadena de cuadro vacío de `WHITE_SQUARE` o `BLACK_SQUARE`, dependiendo del valor en `is_white_square`. Una vez que el código termina de procesar este cuadro del tablero, cambia el valor booleano de `is_white_square` al contrario (porque el siguiente cuadro tiene que ser del color opuesto). Hay que volver a cambiar la variable tras acabar una fila y al final del bucle `for` exterior.

Una vez que ha terminado los bucles, la lista `squares` contiene 64 cadenas. Sin embargo, el método de cadena `format()` no toma un argumento de lista simple, sino un argumento de cadena por par de llaves que hay que sustituir. El asterisco `*` junto a los cuadros dice a Python que pase los valores de esta lista como argumentos individuales. Esto es un poco sutil, pero imagina que tienes una lista `spam = ['cat', 'dog', 'rat']`. Si llamas a `print(spam)`, Python imprimirá el valor de lista, junto con sus corchetes, comillas y comas. Sin embargo, llamar a `print(*spam)` equivale a llamar a `print('cat', 'dog', 'rat')`, que simplemente imprime `cat dog rat`.

La función `print_chessboard()` está escrita para funcionar con la estructura de datos específica que estamos usando para representar el tablero de ajedrez: un diccionario de Python con claves de cadenas de cuadro, como `'a8'`, y valores de cadenas de pieza, como `'bQ'`. Si hubiésemos diseñado nuestra estructura de datos de otra manera, también tendríamos que escribir la función de otra forma. `print_chessboard()` imprime una representación del tablero basada en texto, pero, si estuviésemos usando una librería gráfica, como `Pygame`, para dibujar el tablero, podríamos seguir usando este diccionario de Python para representar la configuración del tablero de ajedrez.

Ahora que tenemos una forma de representar un tablero de ajedrez en un diccionario de Python y una función para mostrar el tablero basado en ese diccionario, vamos a escribir código que mueva las piezas por encima manipulando las claves y los valores del diccionario. Después del bloque def de la función `print_chessboard()`, la parte principal del programa muestra texto que explica cómo usar el programa del tablero interactivo:

```
print('Interactive Chessboard')

print('by Al Sweigart al@inventwithpython.com')

print()

print('Pieces:')

print(' w - White, b - Black')

print(' P - Pawn, N - Knight, B - Bishop, R - Rook, Q - Queen, K - King')

print('Commands:')

print(' move e2 e4 - Moves the piece at e2 to e4')

print(' remove e2 - Removes the piece at e2')

print(' set e2 wP - Sets square e2 to a white pawn')

print(' reset - Resets pieces back to their starting squares')

print(' clear - Clears the entire board')

print(' fill wP - Fills entire board with white pawns.')

print(' quit - Quits the program')
```

El programa puede mover piezas, quitarlas, configurar cuadros con una pieza, restablecer el tablero y limpiarlo cambiando el diccionario del tablero:

```
main_board = copy.copy(STARTING_PIECES)
```

while True:

```
print_chessboard(main_board)
```

```
response = input('> ').split()
```

Para empezar, la variable `main_board` recibe una copia del diccionario `STARTING_PIECES`, que es un diccionario de todas las piezas de ajedrez en sus posiciones iniciales. La ejecución entra en un bucle infinito que permite al usuario introducir comandos. Por ejemplo, si el usuario introduce `move e2 e4` después de llamar a `input()`, el método `split()` devuelve la lista `list ['move', 'e2', 'e4']`, que el programa almacena entonces en la variable `response`. El primer elemento de la lista `response`, `response[0]`, será el comando que quiere ejecutar el usuario:

```
if response[0] == 'move':
```

```
    main_board[response[2]] = main_board[response[1]]
```

```
    del main_board[response[1]]
```

Si el usuario introduce algo como `move e2 e4`, `response[0]` es `'move'`. Podemos “mover” una pieza de un cuadro a otro copiando primero la pieza en `main_board` desde el cuadro anterior (en `response[1]`) al nuevo cuadro (en `response[2]`). Después, podemos borrar el par clave-valor del cuadro antiguo en `main_board`. El efecto es que parece que la pieza se ha movido (aunque no veremos el cambio hasta que volvamos a llamar a `print_chessboard()`).

Nuestro simulador de tablero interactivo no comprueba si el movimiento es válido, simplemente ejecuta las órdenes del usuario. Si el usuario introduce algo como `remove e2`, el programa establecerá `response` en `['remove', 'e2']`:

```
elif response[0] == 'remove':
```

```
    del main_board[response[1]]
```

Al borrar el par clave-valor de la clave `response[1]` de `main_board`, hacemos desaparecer la pieza del tablero. Si el usuario introduce `set e2 wP` para añadir un peón blanco a e2, el programa establecerá

response en ['set', 'e2', 'wP':

elif response[0] == 'set':

main_board[response[1]] = response[2]

Podemos crear un nuevo par clave-valor con la clave response[1] y el valor response[2] en main_board para añadir esta pieza al tablero. Si el usuario introduce reset, response es simplemente ['reset', y podemos devolver el tablero a su configuración original copiando el diccionario STARTING_PIECES en main_board:

elif response[0] == 'reset':

main_board = copy.copy(STARTING_PIECES)

Si el usuario introduce clear, response es ['clear', y podemos quitar todas las piezas de tablero estableciendo main_board como un diccionario vacío:

elif response[0] == 'clear':

main_board = {}

Si el usuario introduce fill wP, response es ['fill', 'wP', y podemos cambiar los 64 cuadros a la cadena 'wP':

elif response[0] == 'fill':

for y in '87654321':

for x in 'abcdefgh':

main_board[x + y] = response[1]

Los bucles for anidados pasan por todos los cuadros, estableciendo la clave x + y en response[1]. No hay una razón real para poner 64 peones blancos en un tablero de ajedrez, pero este comando demuestra lo fácil que es manipular la estructura de datos del tablero de ajedrez como queramos. Por último, el usuario puede salir el programa escribiendo quit:


```
elif response[0] == 'quit':
```

```
sys.exit()
```

Después de ejecutar el comando y modificar `main_board`, la ejecución vuelve al principio del bucle `while` para mostrar el tablero cambiado y aceptar una nueva orden del usuario.

Este programa de tablero de ajedrez interactivo no restringe las piezas que podemos colocar o mover. Simplemente usa un diccionario como representación de las piezas en un tablero y tiene una función para mostrar esos diccionarios en la pantalla de manera que parezcan un tablero de ajedrez. Podemos modelar todos los objetos y procesos del mundo real diseñando estructuras de datos y escribiendo funciones que funcionen con dichas estructuras. Si quieres ver otro ejemplo de modelar un tablero de juego con estructuras de datos, mi otro libro, *The Big Book of Small Python Projects* (No Starch Press, 2021), tiene un programa de tres en raya operativo.

Diccionarios y listas anidados

Al modelar cosas más complicadas, es posible que necesites diccionarios y listas que contengan otros diccionarios y listas. Las listas son útiles para guardar una serie de valores ordenados y los diccionarios para asociar claves con valores. Como ejemplo, aquí tenemos un programa que usa un diccionario para contener diccionarios de artículos que los comensales van a llevar a un picnic. La función `total_brought()` puede leer esta estructura de datos y calcular el número de cada tipo de artículo. Escribe el siguiente código en un programa nuevo y guárdalo como `guestpicnic.py`:

```
all_guests = {'Alice': {'apples': 5, 'pretzels': 12},
```

```
'Bob': {'ham sandwiches': 3, 'apples': 2},
```

```
'Carol': {'cups': 3, 'apple pies': 1}}
```

```
def total_brought(guests, item):
```

```
num_brought = 0

1 for k, v in guests.items():
2     num_brought = num_brought + v.get(item, 0)

return num_brought

print('Number of things being brought:')

print(' - Apples ' + str(total_brought(all_guests, 'apples'))))

print(' - Cups ' + str(total_brought(all_guests, 'cups'))))

print(' - Cakes ' + str(total_brought(all_guests, 'cakes'))))

print(' - Ham Sandwiches ' + str(total_brought(all_guests, 'ham
sandwiches'))))

print(' - Apple Pies ' + str(total_brought(all_guests, 'apple pies'))))
```

Dentro de la función `total_brought()`, el bucle `for` itera por los pares clave-valor de `guests` 1. Dentro del bucle, se asigna a `k` la cadena del nombre del comensal y el diccionario de los artículos que van a llevar al picnic se asigna a `v`. Si el parámetro del artículo existe como clave en este diccionario, su valor (la cantidad) se añade a `num_brought` 2. Si no existe como clave, el método `get()` devuelve 0 para añadir a `num_brought`.

La salida de este programa es así:

Number of things being brought:

- Apples 7
- Cups 3
- Cakes 0
- Ham Sandwiches 3
- Apple Pies 1

El número de artículos llevados a un picnic puede parecer una cosa tan fácil de modelar que no merece la pena molestarse en escribir un programa para hacerlo. Pero date cuenta de que esta misma función `total_brought()` podría manejar fácilmente un diccionario que contenga miles de comensales que lleven miles de artículos diferentes cada uno. En ese caso, tener esa información en una estructura de datos, junto con la función `total_brought()`, ¡te ahorraría un montón de tiempo!

Puedes modelar cosas con estructuras de datos de la forma que quieras, siempre y cuando el resto del código de tu programa pueda funcionar correctamente con el modelo de datos. Cuando empieces a programar, no te preocupes demasiado por la forma “correcta” de modelar datos. A medida que cojas experiencia, puede que des con modelos más eficientes; lo importante es que el modelo de datos funcione para las necesidades de tu programa.

Resumen

En este capítulo, has conocido los diccionarios. Las listas y los diccionarios son valores que pueden contener múltiples valores, incluidos otros diccionarios y listas. Los diccionarios son útiles porque permiten asignar un elemento (la clave) a otro elemento (el valor), mientras que las listas solo contienen una serie de valores en orden. El código puede acceder a los valores dentro de un diccionario usando corchetes, igual que con las listas. En lugar de índices enteros, los diccionarios pueden tener claves de distintos tipos de datos: enteros, flotantes, cadenas o tuplas. Al organizar los valores de un programa en estructuras de datos, puedes crear representaciones de objetos del mundo real como el tablero de ajedrez de este capítulo.

Preguntas prácticas

1. ¿Qué aspecto tiene el código de un diccionario vacío?
2. ¿Qué aspecto tiene un valor de diccionario con ‘foo’ como clave y 42 como valor?
3. ¿Cuál es la principal diferencia entre un diccionario y una lista?

4.¿Qué pasa si intentas acceder a spam['foo' si spam es {'bar': 100}]?

5.Si un diccionario está almacenado en spam, ¿cuál es la diferencia entre las expresiones 'cat' in spam y 'cat' in spam.keys()?

6.Si un diccionario está almacenado en spam, ¿cuál es la diferencia entre las expresiones 'cat' in spam y 'cat' in spam.values()?

7.¿Cómo se puede abreviar el siguiente código?

if 'color' not in spam:

spam['color'] = 'black'

8.¿Qué función de módulo podemos usar para imprimir de forma “bonita” valores de diccionario?

Programas de práctica

Para practicar, escribe programas que realicen las siguientes tareas.

Validar el diccionario de ajedrez

En este capítulo, hemos usado el valor de diccionario {'h1': 'bK', 'c6': 'wQ', 'g2': 'bB', 'h5': 'bQ', 'e3': 'wK'} para representar un tablero de ajedrez. Escribe una función llamada ValidChessBoard() que tome un argumento de diccionario y devuelva True o False dependiendo de si el tablero es válido.

Un tablero válido tendrá exactamente un rey blanco y un rey negro. Cada jugador puede tener, como máximo, 16 piezas, de las cuales solo ocho pueden ser peones, y todas las piezas deben estar en un cuadro válido de '1a' a '8h'. Es decir, una pieza no puede estar en el cuadro '9z'. Los nombres de las piezas deberían empezar por 'w' o 'b' para representar si son blancas o negras, seguido de las mayúsculas para los tipos de ficha que hemos visto antes. Esta función debería detectar si un fallo ha producido un tablero incorrecto. (No es una lista exhaustiva de requisitos, pero para este ejercicio ya está bien).

Inventario de juego de fantasía

Supongamos que estás creando un videojuego de fantasía medieval. La estructura de datos para modelar el inventario del jugador es un diccionario cuyas claves son cadenas que describen el artículo del inventario y cuyos valores son enteros que detallan cuántos artículos de cada tipo tiene el jugador. Por ejemplo, el valor de diccionario {'rope': 1, 'torch': 6, 'gold coin': 42, 'dagger': 1, 'arrow': 12} significa que el jugador tiene una cuerda, seis antorchas, 42 monedas de oro, una daga y doce flechas.

Escribe una función llamada `display_inventory()` que coja cualquier “inventario” posible y lo muestre así:

Inventory:

12 arrow

42 gold coin

1 rope

6 torch

1 dagger

Total number of items: 62

Pista: puedes usar un bucle `for` para pasar por todas las claves de un diccionario.

```
stuff = {'rope': 1, 'torch': 6, 'gold coin': 42, 'dagger': 1, 'arrow': 12}
```

```
def display_inventory(inventory):
```

```
    print("Inventory:")
```

```
    item_total = 0
```

```
    for k, v in inventory.items():
```

```
        # RELLENA ESTA PARTE
```

```
    print("Total number of items: " + str(item_total))
```

```
display_inventory(stuff)
```

Conversión del botín de lista a diccionario

Imagina que el mismo videojuego de fantasía representa el botín de un dragón derrotado como una lista de cadenas:

```
dragon_loot = ['gold coin', 'dagger', 'gold coin', 'gold coin', 'ruby']
```

Escribe una función llamada `add_to_inventory(inventory, added_items)`. El parámetro `inventory` es un diccionario que representa el inventario del jugador (como en el proyecto anterior) y el parámetro `added_items` es una lista, como `dragon_loot`. La función `add_to_inventory()` debería devolver un diccionario que represente el inventario actualizado del jugador. Observa que la lista `added_items` puede contener varios artículos iguales. Tu código debería ser más o menos así:

```
def add_to_inventory(inventory, added_items):
```

```
# Aquí va tu código.
```

```
inv = {'gold coin': 42, 'rope': 1}
```

```
dragon_loot = ['gold coin', 'dagger', 'gold coin', 'gold coin', 'ruby']
```

```
inv = add_to_inventory(inv, dragon_loot)
```

```
display_inventory(inv)
```

El programa (con tu función `display_inventory()` del proyecto anterior) debería mostrar lo siguiente:

Inventory:

45 gold coin

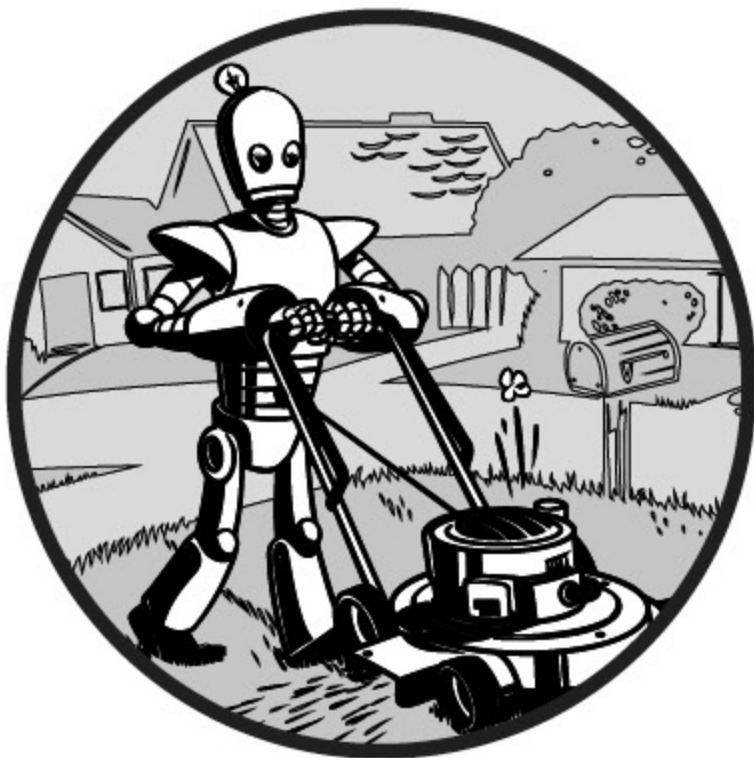
1 rope

1 ruby

1 dagger

Total number of items: 48

Edición de cadenas y texto



El texto es una de las formas de datos más habituales que manejarán tus programas. Ya sabes cómo concatenar dos valores de cadena con el operador `+`, pero podemos hacer mucho más que eso, como extraer cadenas parciales de valores de cadena, añadir o quitar espacios, poner letras en mayúscula o minúscula y comprobar si una cadena tiene el formato correcto. Podemos incluso escribir código Python para acceder al portapapeles que usamos para copiar y pegar texto.

En este capítulo, aprenderás todo esto y más. Después, trabajaremos en un proyecto de programación para automatizar la aburrida tarea

de añadir viñetas a un texto.

Trabajar con cadenas

Echemos un vistazo a algunas de las maneras en las que Python nos permite escribir, imprimir y acceder a cadenas en nuestro código.

Cadenas literales

Aunque los valores de cadena se guardan en la memoria del programa, los valores de cadena que aparecen literalmente en el código se llaman “cadenas literales”. Escribir cadenas literales en código Python parece fácil: empiezan y terminan con una comilla simple y llevan el texto del valor de cadena entre medio. ¿Pero cómo podemos usar comillas dentro de una cadena? Escribir ‘That is Alice’s cat.’ no funcionará porque Python pensará que la cadena termina después de Alice y tratará el resto (s cat.) como código no válido. Por suerte hay varias formas de escribir cadenas literales. El término “cadena” se refiere a un único valor en el contexto de un programa en ejecución y a una cadena literal cuando estamos hablando de escribir código fuente en Python.

Comillas dobles

Las cadenas literales pueden empezar y terminar con comillas dobles, igual que con comillas simples. Una ventaja de usar las dobles es que la cadena puede contener una comilla simple dentro. Escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > spam = "That is Alice's cat."
```

Como la cadena empieza y acaba con comillas dobles, Python sabe que la comilla simple forma parte de la cadena y no la corta ahí. No obstante, si necesitas usar comillas simples y dobles dentro de la cadena, tendrás que recurrir a los caracteres de escape.

Caracteres de escape

Un carácter de escape permite usar caracteres que de otro modo

sería imposible poner en una cadena literal. Un carácter de escape consiste en una barra invertida (\) seguida del carácter que se quiere añadir a la cadena. Por ejemplo, \' es el carácter de escape para una comilla simple y \n es el carácter de escape para el carácter de salto de línea. (Pese a consistir en dos caracteres, a menudo se usa el nombre carácter de escape en singular). Puedes usar esta sintaxis dentro de una cadena que empieza y termina con comillas simples. Para ver cómo funcionan los caracteres de escape, escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > spam = 'Say hi to Bob\'s mother.'
```

Python sabe que, como la comilla de Bob\'s lleva una barra invertida, no es una comilla que pretenda terminar el valor de cadena. Los caracteres de escape \' y \" permiten poner comillas simples y dobles, respectivamente, dentro de una cadena.

La tabla 8.1 recoge los caracteres de escape que puedes usar.

Tabla 8.1. Caracteres de escape.

Carácter de escape.

Comilla simple

Comillas dobles

Tabulación

Salto de línea

Barra invertida

Para practicar con esto, escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > print("Hello there!\nHow are you?\nI\'m doing fine.")
```

Hello there!

How are you?

I'm doing fine.

Ten en cuenta que como \ inicia un carácter de escape, si necesitas poner una barra invertida en tu cadena, debes usar el carácter de escape \\.

Cadenas crudas

Puedes poner una `r` antes de la comilla inicial de una cadena para marcarla como cadena literal cruda. Una cadena cruda hace que sea más fácil introducir valores de cadena que contengan barras invertidas ignorando todos los caracteres de escape. Para ver un ejemplo, pon esto en el intérprete de comandos interactivo:

```
> > > print(r'The file is in C:\Users\Alice\Desktop')
```

The file is in C:\Users\Alice\Desktop

Como se trata de una cadena cruda, Python considera que la barra invertida es parte de la cadena y no el inicio de un carácter de escape:

```
> > > print('Hello...\n\n...world!') # Sin cadena cruda.
```

Hello...

...world!

```
> > > print(r'Hello...\n\n...world!') # Con cadena cruda.
```

Hello...\n\n...world!

Las cadenas crudas son útiles si los valores de cadena contienen varias barras invertidas, como las utilizadas para las rutas de archivo en Windows, como `r'C:\Users\Al\Desktop'`, o expresiones regulares, que veremos en el siguiente capítulo.

Cadenas multilínea

Aunque puedes usar el carácter de escape `\n` para insertar un salto de línea en una cadena, suele ser más fácil usar cadenas multilínea. Una cadena multilínea de Python empieza y termina con tres comillas simples o dobles. Cualquier comilla, tabulación o salto de línea que quede entre las “comillas triples” se considera parte de la cadena. Las reglas de sangrado de Python para los bloques no se aplican a las líneas de una cadena multilínea.

Como ejemplo, abre el editor de archivos y escribe lo siguiente:

```
print("""Dear Alice,  
  
Can you feed Eve's cat this weekend?  
  
Sincerely,  
  
Bob""")
```

Guarda el programa como `feedcat.py` y ejecútalo. La salida será así:

```
Dear Alice,  
  
Can you feed Eve's cat this weekend?  
  
Sincerely,  
  
Bob
```

Observa que el carácter de comilla simple en `Eve's` no requiere escape. Los caracteres de escape y las comillas dobles son opcionales en las cadenas multilínea:

```
print("Dear Alice,\n\nCan you feed Eve's cat this weekend?\n\nSincerely,\nBob")
```

Esta llamada a `print()` imprime el mismo texto exactamente, pero no usa una cadena multilínea.

Comentarios multilínea

Aunque la almohadilla (`#`) marca el inicio de un comentario para el resto de la línea, a menudo se usa una cadena multilínea para comentarios que se extienden varias líneas:

```
"""Esto es un programa de prueba de Python.
```

Escrito por Al Sweigart al@inventwithpython.com

Este programa se diseñó para Python 3, no Python 2.

“””

```
def say_hello():
```

```
    """Esta función imprime hello.
```

```
    No devuelve nada."""
```

```
print('Hello!')
```

La cadena multilínea de este ejemplo es código Python perfectamente válido.

Índices y rebanadas

Las cadenas utilizan índices y rebanadas del mismo modo que las listas. Imagina la cadena ‘Hello, world!’ como una lista; cada carácter de la cadena es un elemento con un índice e índice negativo correspondiente:

H
o
-o

El espacio y la exclamación están incluidos en la cuenta de caracteres, así que ‘Hello, world!’ tiene 13 caracteres, desde H en el índice 0 hasta ! en el índice 12.

Escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > greeting = 'Hello, world!'
```

```
> > > greeting[0]
```

```
'H'
```

```
> > > greeting [4]
```

```
'o'
```

```
> > > greeting[-1]
```

'!'

```
> > > greeting[0:5]
```

'Hello'

```
> > > greeting[:5]
```

'Hello'

```
> > > greeting[7:-1]
```

'world'

```
> > > greeting[7:]
```

'world!'

Si especificamos un índice, obtendremos el carácter que ocupa esa posición en la cadena. Si especificamos un rango de un índice a otro, se incluye el índice inicial pero no el final. Por eso, si el saludo es 'Hello, world!', `greeting[0:5]` se evalúa como 'Hello'. La subcadena que obtenemos de `greeting[0:5]` incluye todo entre `greeting[0]` y `greeting[4]`, dejando fuera la coma del índice 5 y el espacio del índice 6. Esto es parecido a cómo `range(5)` hará que un bucle `for` itere hasta, pero no incluya, 5.

Observa que hacer rebanadas en una cadena no modifica la cadena original. Podemos hacer una rebanada de una variable en otra variable. Prueba a introducir lo siguiente en el intérprete de comandos interactivo:

```
> > > greeting = 'Hello, world!'
```

```
> > > greeting_slice = greeting[0:5]
```

```
> > > greeting_slice
```

'Hello'

```
> > > greeting
```

'Hello, world!'

Al hacer una rebanada y guardar la subcadena resultante en otra variable, podemos tener la cadena completa y la subcadena a mano para un acceso rápido y sencillo.

Los operadores `in` y `not in`

Podemos usar los operadores `in` y `not in` con cadenas igual que con valores de lista. Una expresión con dos cadenas unida con `in` o `not in` se evaluará a un valor booleano `True` o `False`. Escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > 'Hello' in 'Hello, World'
```

`True`

```
> > > 'Hello' in 'Hello'
```

`True`

```
> > > 'HELLO' in 'Hello, World'
```

`False`

```
> > > ' ' in 'spam'
```

`True`

```
> > > 'cats' not in 'cats and dogs'
```

`False`

Estas expresiones comprueban si la primera cadena (teniendo en cuenta las mayúsculas) se encuentra dentro de la segunda cadena.

Cadenas formateadas

Poner cadenas dentro de otras cadenas es una operación habitual en programación. Hasta ahora, hemos estado usando el operador `+` y la concatenación de cadenas para hacerlo:

```
> > > name = 'Al'
```

```
> > > age = 4000
```

```
> > > 'Hello, my name is ' + name + '. I am ' + str(age) + ' years old.'
```

```
'Hello, my name is Al. I am 4000 years old.'
```

```
> > > 'In ten years I will be ' + str(age + 10)
```

```
'In ten years I will be 4010'
```

Sin embargo, esto requiere escribir mucho. Un enfoque más sencillo es usar cadenas formateadas, o *f-strings*, lo que nos permite colocar nombres de variables o expresiones enteras dentro de una cadena. Al igual que el prefijo *r* en las cadenas crudas, las cadenas formateadas llevan un prefijo *f* antes de la comilla inicial. Introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > name = 'Al'
```

```
> > > age = 4000
```

```
> > > f'My name is {name}. I am {age} years old.'
```

```
'My name is Al. I am 4000 years old.'
```

```
> > > f'In ten years I will be {age + 10}'
```

```
'In ten years I will be 4010'
```

Todo lo que va entre las llaves (*{}*) se interpreta como si se pasase a *str()* y se concatenase con el operador *+* en medio de la cadena. Si necesitas usar llaves literales en una *f-string*, usa dos llaves:

```
> > > name = 'Zophie'
```

```
> > > f'{name}'
```

```
'Zophie'
```

```
> > > f'{{name}}' # Las llaves dobles son llaves literales.
```

```
'{name}'
```


Las cadenas formateadas son una característica útil de Python, pero el lenguaje las incorporó en la versión 3.6. En código Python anterior, puede que te encuentres con técnicas alternativas.

Alternativas a las cadenas formateadas: %s y format()

Las versiones de Python anteriores a la 3.6 tenían otras formas de poner cadenas dentro de otras cadenas. La primera era la interpolación de cadenas, donde las cadenas incluían un especificador de formato %s que Python reemplazaba con otra cadena. Para ver un ejemplo, escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > name = 'Al'
> > > age = 4000
> > > 'My name is %s. I am %s years old.' % (name, age)
'My name is Al. I am 4000 years old.'
> > > 'In ten years I will be %s' % (age + 10)
'In ten years I will be 4010'
```

Python sustituirá el primer %s por el primer valor en los paréntesis después de la cadena, el segundo %s por la segunda cadena y así sucesivamente. Esto funciona igual de bien que las cadenas formateadas si solo necesitas insertar una o dos cadenas, pero las *f-strings* son más legibles cuando hay muchas cadenas que insertar.

La otra técnica para poner cadenas dentro de otras cadenas es usar el método de cadena format(). Puedes usar un par de llaves para marcar dónde insertar la cadena, igual que en la interpolación. Escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > name = 'Al'
> > > age = 4000
> > > 'My name is . I am } years old.'.format(name, age)
```

'My name is Al. I am 4000 years old.'

El método `format()` tiene unas cuantas prestaciones más que la interpolación con `%s`. Podemos poner el entero del índice (empezando en 0) dentro de las llaves para anotar cuál de los argumentos de `format()` debería insertarse. Esto es útil cuando se insertan cadenas varias veces o sin orden:

```
> > > name = 'Al'
```

```
> > > age = 4000
```

```
> > > '{1} years ago, {0} was born and named  
{0}.'.format(name, age)
```

'4000 years ago, Al was born and named Al.'

La mayoría de los programadores prefieren las cadenas formateadas a estas dos alternativas, pero te conviene saberlas de todos modos por si te las encuentras en código antiguo.

Métodos de cadena útiles

Hay varios métodos de cadena para analizar cadenas o crear valores de cadena transformados. Este apartado describe los métodos que usarás con mayor frecuencia.

Cambiar mayúsculas y minúsculas

Los métodos de cadena `upper()` y `lower()` devuelven una nueva cadena con todas las letras de la original convertidas a mayúscula o minúscula, respectivamente. Los caracteres de la cadena que no son letras se quedan igual. Por ejemplo, escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > spam = 'Hello, world!'
```

```
> > > spam = spam.upper()
```

```
> > > spam
```

`'HELLO, WORLD!'`

```
> > > spam = spam.lower()
```

```
> > > spam
```

`'hello, world!'`

Observa que estos métodos no cambian la cadena en sí, sino que devuelven nuevos valores de cadena. Si quieres cambiar la cadena original, tienes que llamar a `upper()` o `lower()` con la cadena y asignar la nueva cadena a la variable que almacenaba la original. Por eso debes usar `spam = spam.upper()` para cambiar la cadena en vez de escribir simplemente `spam.upper()`. (Es lo mismo que si una variable `eggs` contuviese el valor 10. Escribir `eggs + 3` no cambia el valor de `eggs`, pero `eggs = eggs + 3` sí).

Los métodos `upper()` y `lower()` son útiles si necesitas hacer una comparación en la que no importen las mayúsculas o minúsculas. Por ejemplo, las cadenas `'great'` y `'GREAt'` no son iguales entre sí, pero, en el siguiente programa, el usuario puede escribir `Great`, `GREAT` o `grEAT`, porque el código convertirá la cadena a minúsculas:

```
print('How are you?')
```

```
feeling = input()
```

```
if feeling.lower() == 'great':
```

```
    print('I feel great too.')
```

```
else:
```

```
    print('I hope the rest of your day is good.')
```

Al ejecutar este programa muestra una pregunta y escribir cualquier variación de `great`, como `GREAt`, producirá la salida `I feel great too`. Añadir código para que gestione variaciones o errores en la entrada del usuario, como un uso inconsistente de mayúsculas, hará tus programas más fáciles de usar y menos propensos a fallar:

How are you?

GREat

I feel great too.

Los métodos `isupper()` e `islower()` devolverán un valor booleano `True` si la cadena contiene al menos una letra y todas las letras son mayúsculas o minúsculas, respectivamente. En caso contrario, el método devuelve `False`. Escribe lo siguiente en el intérprete de comandos interactivo y observa lo que devuelve cada método:

```
> > > spam = 'Hello, world!'
```

```
> > > spam.islower()
```

False

```
> > > spam.isupper()
```

False

```
> > > 'HELLO'.isupper()
```

True

```
> > > 'abc12345'.islower()
```

True

```
> > > '12345'.islower()
```

False

```
> > > '12345'.isupper()
```

False

Como los métodos de cadena `upper()` y `lower()` devuelven cadenas, también podemos llamar a métodos de cadena con los valores que devuelven:

```
> > > 'Hello'.upper()
```

```
'HELLO'
```

```
> > > 'Hello'.upper().lower()
```

```
'hello'
```

```
> > > 'Hello'.upper().lower().upper()
```

```
'HELLO'
```

```
> > > 'HELLO'.lower()
```

```
'hello'
```

```
> > > 'HELLO'.lower().islower()
```

```
True
```

Las expresiones que hacen esto parecerán una cadena de llamadas a métodos.

Comprobar las características de una cadena

Además de `islower()` e `isupper()`, hay otros métodos de cadena cuyos nombres empiezan por `is`. Estos métodos devuelven un valor booleano que describe la naturaleza de la cadena. Estos son algunos métodos de cadena `isX()` habituales:

- `isalpha()`: Devuelve `True` si la cadena consta solo de letras y no está vacía.

- `isalnum()`: Devuelve `True` si la cadena consta solo de letras y números y no está vacía.

- `isdecimal()`: Devuelve `True` si la cadena consta solo de caracteres numéricos y no está vacía.

- `isspace()`: Devuelve `True` si la cadena consta solo de espacios, tabulaciones y saltos de línea y no está vacía.

- `istitle()`: Devuelve `True` si la cadena consta solo de palabras que empiezan por mayúscula seguida solo por letras minúsculas.

Introduce esto en el intérprete de comandos interactivo:

```
> > > 'hello'.isalpha()
```

True

```
> > > 'hello123'.isalpha()
```

False

```
> > > 'hello123'.isalnum()
```

True

```
> > > 'hello'.isalnum()
```

True

```
> > > '123'.isdecimal()
```

True

```
> > > ' '.isspace()
```

True

```
> > > 'This Is Title Case'.istitle()
```

True

Los métodos de cadena isX() son útiles cuando necesitamos validar la entrada del usuario. Por ejemplo, el siguiente programa pide repetidamente al usuario su edad y contraseña hasta que proporciona una entrada válida. Abre una ventana nueva en el editor de archivos, escribe este programa y guárdalo como validateInput.py:

```
while True:
```

```
    print('Enter your age:')
```

```
    age = input()
```

```
if age.isdecimal():  
  
    break  
  
    print('Please enter a number for your age.')  
while True:  
  
    print('Select a new password (letters and numbers only):')  
    password = input()  
  
    if password.isalnum():  
  
        break  
  
    print('Passwords can only have letters and numbers.')
```

En el primer bucle while, pedimos al usuario su edad y almacenamos su respuesta en age. Si age es un valor (decimal) válido, salimos de este primer bucle y pasamos al siguiente, que pide una contraseña. De lo contrario, informamos al usuario de que debe introducir un número y volvemos a preguntarle la edad. En el segundo bucle while, pedimos una contraseña, guardamos la respuesta del usuario en password y salimos del bucle si la entrada es alfanumérica. Si no lo es, no estamos satisfechos, así que decimos al usuario que la contraseña debe ser alfanumérica y volvemos a pedirle que introduzca una contraseña. Cuando ejecutamos el programa, la salida tiene este aspecto:

Enter your age:

forty two

Please enter a number for your age.

Enter your age:

42

Select a new password (letters and numbers only):

secr3t!

Passwords can only have letters and numbers.

Select a new password (letters and numbers only):

secr3t

Llamando a `isdecimal()` e `isalnum()` con variables, podemos determinar si los valores almacenados en esas variables son decimales o no y alfanuméricos o no. Aquí, estas pruebas nos ayudan a rechazar la entrada `forty two` pero aceptar `42`, y rechazar `secr3t!` pero aceptar `secr3t`.

Comprobar el principio o el final de una cadena

Los métodos `startswith()` y `endswith()` devuelven `True` si el valor de la cadena con la que se les ha llamado empieza o termina, respectivamente, con la cadena pasada al método; si no es el caso, devuelven `False`. Escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > 'Hello, world!'.startswith('Hello')
```

`True`

```
> > > 'Hello, world!'.endswith('world!')
```

`True`

```
> > > 'abc123'.startswith('abcdef')
```

`False`

```
> > > 'abc123'.endswith('12')
```

`False`

```
> > > 'Hello, world!'.startswith('Hello, world!')
```

`True`

```
> > > 'Hello, world!'.endswith('Hello, world!')
```


True

Estos métodos son alternativas útiles al operador de igualdad (`==`) cuando solo necesitamos comprobar si la primera o la última parte de una cadena, y no la cadena entera, es igual que otra cadena.

Unir y separar cadenas

El método `join()` es útil cuando tenemos una lista de cadenas que hay que unir en un único valor de cadena. Llamamos a `join()` con una cadena, le pasamos una lista de cadenas y devuelve una concatenación de las cadenas de la lista que le hemos pasado. Para ver un ejemplo, introduce esto en el intérprete de comandos interactivo:

```
> > > , '.join(['cats', 'rats', 'bats'])
```

```
'cats, rats, bats'
```

```
> > > ' '.join(['My', 'name', 'is', 'Simon'])
```

```
'My name is Simon'
```

```
> > > 'ABC'.join(['My', 'name', 'is', 'Simon'])
```

```
'MyABCnameABCisABCSimon'
```

Observa que la cadena con la que llamamos a `join()` está insertada entre las cadenas del argumento de lista. Por ejemplo, si llamamos a `join(['cats', 'rats', 'bats'])` con la cadena `' '`, devuelve la cadena `'cats, rats, bats'`.

Recuerda que podemos llamar a `join()` con un valor de cadena y pasarle un valor de lista. (Es fácil llamarlo al revés por error). El método `split()` funciona al revés: lo llamamos con un valor de cadena y devuelve una lista de cadenas. Escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > 'My name is Simon'.split()
```

```
['My', 'name', 'is', 'Simon']
```

Por defecto, el método divide la cadena 'My name is Simon' donde encuentra un espacio en blanco, como un espacio, una tabulación o un salto de línea. Estos caracteres de espacio en blanco no están incluidos en las cadenas de la lista devuelta. Podemos pasar una cadena delimitadora al método `split()` para especificar una cadena diferente para la descomposición. Por ejemplo, escribe esto en el intérprete de comandos interactivo:

```
> > > 'MyABCnameABCisABCSimon'.split('ABC')
```

```
['My', 'name', 'is', 'Simon']
```

```
> > > 'My name is Simon'.split('m')
```

```
['My na', 'e is Si', 'on']
```

Un uso habitual de `split()` es dividir una cadena multilínea a través de los saltos de línea. Para ver un ejemplo, introduce esto en el intérprete de comandos interactivo:

```
> > > spam = '''Dear Alice,
```

```
. There is a milk bottle in the fridge
```

```
. that is labeled "Milk Experiment."
```

```
...
```

```
. Please do not drink it.
```

```
. Sincerely,
```

```
. Bob'''
```

```
.
```

```
> > > spam.split('\n')
```

```
['Dear Alice,', 'There is a milk bottle in the fridge',
```

```
'that is labeled "Milk Experiment."', ', 'Please do not drink it.',
```

```
'Sincerely,', 'Bob']
```

Pasar a `split()` el argumento `'\n'` nos permite descomponer la cadena multilínea almacenada en `spam` a través de los saltos de línea y devuelve una lista en la que cada elemento se corresponde con una línea de la cadena.

Justificar y centrar texto

Los métodos de cadena `rjust()` y `ljust()` devuelven una versión rellena de la cadena con la que se los llama, con espacios en blanco insertados para justificar el texto. El primer argumento de ambos métodos es un entero para la longitud de la cadena justificada. Escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > 'Hello'.rjust(10)
```

```
' Hello'
```

```
> > > 'Hello'.rjust(20)
```

```
' Hello'
```

```
> > > 'Hello, World'.rjust(20)
```

```
' Hello, World'
```

```
> > > 'Hello'.ljust(10)
```

```
'Hello '
```

El código `'Hello'.rjust(10)` dice que queremos justificar a la derecha `'Hello'` en una cadena de una longitud total de 10. `'Hello'` tiene cinco caracteres, así que se añadirán cinco espacios a la izquierda, lo que nos dará una cadena de 10 caracteres, con `'Hello'` justificado a la derecha.

Un segundo argumento opcional para `rjust()` y `ljust()` especificará un carácter de relleno distinto del espacio. Escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > 'Hello'.rjust(20, )
```

```
*****Hello'
```

```
> > > 'Hello'.ljust(20, )
```

```
'Hello-----'
```

El método de cadena `center()` funciona igual que `ljust()` y `rjust()`, pero centra el texto en vez de justificarlo a la izquierda o a la derecha. Escribe esto en el intérprete de comandos interactivo:

```
> > > 'Hello'.center(20)
```

```
' Hello '
```

```
> > > 'Hello'.center(20, )
```

```
'=====Hello====='
```

Ahora el texto que se muestra está centrado.

Eliminar espacios en blanco

A veces, puede que necesitemos quitar caracteres de espacio en blanco (espacios, tabulaciones y saltos de línea) de la derecha, la izquierda o ambos lados de una cadena. El método `strip()` devolverá una cadena nueva sin espacios en blanco al principio o al final, mientras que los métodos `lstrip()` y `rstrip()` quitarán caracteres de espacio de la izquierda y la derecha respectivamente. Escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > spam = ' Hello, World '
```

```
> > > spam.strip()
```

```
'Hello, World'
```

```
> > > spam.lstrip()
```

```
'Hello, World '
```

```
> > > spam.rstrip()
```

```
'Hello, World'
```

Opcionalmente, un argumento de cadena especificará qué caracteres quitar del final. Introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > spam = 'SpamSpamBaconSpamEggsSpamSpam'
```

```
> > > spam.strip('ampS')
```

```
'BaconSpamEggs'
```

Pasar a `strip()` el argumento `'ampS'` ordenará quitar todas las apariciones de a, m, p y S del final de las cadenas almacenadas en `spam`. No importa el orden de los caracteres en la cadena pasada a `strip()`: `strip('ampS')` hará lo mismo que `strip('mapS')` o `strip('Spam')`.

Puntos de código numéricos de caracteres

Los ordenadores guardan información como bytes (cadenas de números binarios), lo que significa que tenemos que ser capaces de convertir texto a números. Dado este requisito, todo carácter de texto tiene un valor numérico correspondiente llamado punto de código Unicode. Por ejemplo, el punto de código numérico de `'A'` es 65, el de `'4'` es 52 y el `'!'` de es 33. Podemos usar la función `ord()` para obtener el punto de código de una cadena de un carácter y la función `chr()` para obtener la cadena de un carácter de un punto de código entero. Escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > ord('A')
```

```
65
```

```
> > > ord('4')
```

```
52
```

```
> > > ord('!')
```

```
33
```

```
> > > chr(65)
```

```
'A'
```

Estas funciones son útiles cuando se necesita ordenar o realizar una operación matemática con caracteres:

```
> > > ord('B')
```

```
66
```

```
> > > ord('A') < ord('B')
```

```
True
```

```
> > > chr(ord('A'))
```

```
'A'
```

```
> > > chr(ord('A') + 1)
```

```
'B'
```

Unicode y los puntos de código son más complejos que esto, pero esos detalles quedan fuera del ámbito de este libro. Si quieres saber más, te recomiendo ver o leer la charla de Ned Batchelder en PyCon 2012 *Pragmatic Unicode, or How Do I Stop the Pain?*, en <https://nedbatchelder.com/text/unipain.html>.

Cuando se escriben cadenas en un archivo o se envían por Internet, la conversión de texto a bytes se denomina codificación. Hay varios estándares de codificación Unicode, pero el más popular es UTF-8. Si alguna vez tienes que elegir una codificación Unicode, 'utf-8' es la respuesta correcta el 99 por ciento de las veces. Tom Scott tiene un vídeo en Computerphile titulado *Characters, Symbols and the Unicode Miracle*, en <https://www.youtube.com/watch?v=MijmeoH9LT4>, que explica UTF-8 en concreto.

Copiar y pegar cadenas

El módulo `pyperclip` tiene funciones `copy()` y `paste()` que pueden enviar y recibir texto al y del portapapeles del ordenador. Enviar la

salida del programa al portapapeles hará que sea más fácil pegarla en un correo electrónico, un procesador de texto y otro software. El módulo `pyperclip` no viene con Python. Para instalarlo, sigue las instrucciones para instalar paquetes de terceros del apéndice A. Tras instalar `pyperclip`, escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > import pyperclip

> > > pyperclip.copy('Hello, world!')

> > > pyperclip.paste()

'Hello, world!'
```

Naturalmente, si algo fuera del programa cambia el contenido del portapapeles, la función `paste()` devolverá ese otro valor. Por ejemplo, si copiase esta frase en el portapapeles y luego llamase a `paste()`, tendría este aspecto:

```
> > > pyperclip.paste()

'Por ejemplo, si copiase esta frase en el portapapeles y luego
llamase
a paste(), tendría este aspecto:'
```

El portapapeles es una forma excelente de introducir y recibir grandes cantidades de texto sin tener que escribirlo cuando así lo indique una llamada a `input()`. Por ejemplo, supongamos que queremos que un programa convierta texto a una `aLtErNaNcIa` de mayúsculas y minúsculas. Podemos copiar el texto en cuestión en el portapapeles y luego ejecutar este programa, que lo coge para hacer lo de las mayúsculas y minúsculas y lo devuelve al portapapeles. Escribe el siguiente código en un archivo llamado `alternatingText.py`:

```
import pyperclip

text = pyperclip.paste() # Coge el texto del portapapeles.

alt_text = '' # Esta cadena tiene las mayúsculas y minúsculas
```

alternas.

```
make_uppercase = False
```

```
for character in text:
```

```
# Pasa por cada carácter y lo añade a alt_text:
```

```
if make_uppercase:
```

```
alt_text += character.upper()
```

```
else:
```

```
alt_text += character.lower()
```

```
# Establece make_uppercase en el valor opuesto:
```

```
make_uppercase = not make_uppercase
```

```
pyperclip.copy(alt_text) # Pone el resultado en el portapapeles.
```

```
print(alt_text) # Imprime el resultado en la pantalla.
```

Si copias texto en el portapapeles (por ejemplo, esta oración) y ejecutas este programa, la salida y el contenido del portapapeles se convertirá en esto:

sI CoPiAs tExTo eN El pOrtApApEleS (pOr eJeMpLo, EsTa oRaCiÓN)
Y EjEcUtAs

eStE PrOgRaMa, La sAlIdA Y eL CoNtEnIdO DeL PoRtApApEleS Se
cOnVeRtIrÁ En eStO:

La capacidad del módulo pyperclip para interactuar con el portapapeles nos da una manera fácil de introducir y extraer texto en nuestros programas.

Proyecto 2: Añadir viñetas a marcado wiki

Cuando editamos un artículo de Wikipedia, podemos crear una lista de viñetas poniendo cada elemento de la lista en su propia línea y

colocando un asterisco delante. Pero supongamos que tienes una lista realmente larga en la que añadir viñetas. Podrías escribir esos asteriscos al principio de cada línea, uno por uno. O podrías automatizar esta tarea con un pequeño *script* de Python.

El *script* `bulletPointAdder.py` cogerá el texto del portapapeles, añadirá un asterisco y un espacio al principio de cada línea y luego pegará el nuevo texto en el portapapeles. Por ejemplo, supongamos que copiamos el siguiente texto (del artículo de Wikipedia *List of Lists of Lists*) en el portapapeles:

Lists of animals

Lists of aquarium life

Lists of biologists by author abbreviation

Lists of cultivars

Si ejecutásemos el programa `bulletPointAdder.py`, el portapapeles contendría:

- * Lists of animals
- * Lists of aquarium life
- * Lists of biologists by author abbreviation
- * Lists of cultivars

Este texto con el prefijo del asterisco está listo para copiarse en un artículo de Wikipedia como lista de viñetas.

Paso 1: copiar y pegar del portapapeles

Queremos que el programa `bulletPointAdder.py` haga lo siguiente:

- Pegar texto desde el portapapeles.
- Hacer algo con él.
- Devolver el texto al portapapeles.

Manipular el texto es un poco difícil, pero copiarlo y pegarlo es bastante fácil: solo necesitamos las funciones `pyperclip.copy()` y `pyperclip.paste()`. Por ahora, vamos a escribir la parte del programa que llama a estas funciones. Escribe lo siguiente y guarda como `bulletPointAdder.py`:

```
import pyperclip

text = pyperclip.paste()

# PENDIENTE: separar líneas y añadir asteriscos.

pyperclip.copy(text)
```

El comentario PENDIENTE es un recordatorio de que deberías completar esta parte del programa al final. El siguiente paso es implementar realmente ese fragmento.

Paso 2: separar las líneas de texto

La llamada a `pyperclip.paste()` devuelve todo el texto del portapapeles como una gran cadena. Si usásemos el ejemplo del artículo *List of Lists of Lists*, la cadena almacenada en `text` quedaría así:

```
'Lists of animals\nLists of aquarium life\nLists of biologists by
author
abbreviation\nLists of cultivars'
```

Los caracteres de salto de línea `\n` de esta cadena hacen que se muestre con varias líneas impresas o pegadas desde el portapapeles. Hay muchas “líneas” en este valor de cadena. Queremos añadir un asterisco al principio de cada una.

Podríamos escribir código que busque cada carácter `\n` en la cadena y añada el asterisco justo detrás. Pero sería más fácil usar el método `split()` para devolver una lista de cadenas, una por cada línea de la cadena original, y añadir después el asterisco delante de cada cadena de la lista.

Edita tu programa para que quede así:

```
import pyperclip

text = pyperclip.paste()

# Separa líneas y añade asteriscos.

lines = text.split('\n')

for i in range(len(lines)): # Pasa en bucle por todos los índices de
la lista "lines".

lines[i] = * ' + lines[i] # Añade un asterisco a cada cadena de la
lista "lines".

pyperclip.copy(text)
```

Dividimos el texto por saltos de línea nuevos para obtener una lista en la que cada elemento sea una línea de texto. Almacenamos la lista en `lines` y pasamos en bucle por los elementos de `lines`. Para cada línea, añadimos un asterisco y un espacio al principio de la línea. Ahora cada cadena de `lines` empieza con un asterisco.

Paso 3: unir las líneas modificadas

La lista `lines` contiene ahora líneas modificadas que empiezan con asteriscos. Pero `pyperclip.copy()` espera un único valor de cadena, no una lista de valores. Para convertir esto en un único valor, pasamos `lines` al método `join()` para obtener una única cadena unida a partir de las cadenas de la lista:

```
import pyperclip

text = pyperclip.paste()

# Separa líneas y añade asteriscos.

lines = text.split('\n')

for i in range(len(lines)): # Pasa en bucle por todos los índices de la
lista "lines".
```

`lines[i] = '*' + lines[i] # Añade un asterisco a cada cadena de la lista "lines".`

`text = '\n'.join(lines)`

`pyperclip.copy(text)`

Cuando se ejecuta el programa, este sustituye el texto en el portapapeles por texto que tiene asteriscos al principio de cada línea. Ahora que el programa está completo, puedes probar a ejecutarlo con texto copiado en el portapapeles.

Incluso aunque no necesites automatizar esta tarea específica, puede que tengas que hacerlo con otro tipo de manipulación de texto, como quitar espacios al final de líneas o convertir texto a mayúsculas o minúsculas. Siempre que lo necesites, puedes usar el portapapeles para entrada y salida.

Un programa corto: pig latin

El *pig latin* es una tontería de lenguaje inventado que altera palabras en inglés. Si una palabra empieza por vocal, se le añade “yay” al final. Si empieza por consonante o grupo consonántico (como “ch” o “gr”), esa(s) consonante(s) se mueve(n) al final de la palabra y se añade “ay”.

Vamos a escribir un programa *pig latin* que produzca algo como esto:

Enter the English message to translate into pig latin:

My name is AL SWEIGART and I am 4,000 years old.

Ymay amenay isyay ALYAY EIGARTSWAY andyay Iyay amyay
4,000 yearsyay oldyay.

Este programa funciona alterando una cadena con los métodos introducidos en este capítulo. Escribe el siguiente código fuente en el editor de archivos y guárdalo como `pigLat.py`:

`# De inglés a pig latin`

```
print("Enter the English message to translate into pig latin:")

message = input()

VOWELS = ('a', 'e', 'i', 'o', 'u', 'y')

pig_latin = [] # Una lista de las palabras en pig latin

for word in message.split():

    # Separa lo que no son letras al principio de esta palabra:

    prefix_non_letters = ""

    while len(word) > 0 and not word[0].isalpha():

        prefix_non_letters += word[0]

        word = word[1:]

    if len(word) == 0:

        pig_latin.append(prefix_non_letters)

    continue

    # Separa lo que no son letras al final de esta palabra:

    suffix_non_letters = ""

    while not word[-1].isalpha():

        suffix_non_letters = word[-1] + suffix_non_letters

        word = word[:-1]

    # Recuerda si la palabra iba en mayúsculas o con mayúscula inicial:

    was_upper = word.isupper()

    was_title = word.istitle()

    word = word.lower() # Pon la palabra en minúsculas para la
```

traducción.

Separa las consonantes al principio de esta palabra:

```
prefix_consonants = ''
```

```
while len(word) > 0 and not word[0] in VOWELS:
```

```
    prefix_consonants += word[0]
```

```
    word = word[1:]
```

Añade el sufijo de pig latin a la palabra:

```
if prefix_consonants != '':
```

```
    word += prefix_consonants + 'ay'
```

```
else:
```

```
    word += 'yay'
```

Pon otra vez la palabra en mayúsculas o con mayúscula inicial:

```
if was_upper:
```

```
    word = word.upper()
```

```
if was_title:
```

```
    word = word.title()
```

Vuelve a poner lo que no son letras al principio o al final de la palabra.

```
pig_latin.append(prefix_non_letters + word + suffix_non_letters)
```

Vuelve a unir todas las palabras en una única cadena:

```
print(' '.join(pig_latin))
```

Echemos un vistazo a este código, línea por línea, empezando desde arriba:

De inglés a pig latin

```
print('Enter the English message to translate into pig latin:')
```

```
message = input()
```

```
VOWELS = ('a', 'e', 'i', 'o', 'u', 'y')
```

Primero, pedimos al usuario que escriba el texto que hay que traducir a *pig latin*. También creamos una constante que contenga todas las vocales en minúscula (e y) como tupla de cadenas. Usaremos esta variable más adelante.

A continuación, creamos la variable `pigLatin` para guardar las palabras cuando las traduzcamos a *pig latin*:

```
pigLatin = [] # Una lista de las palabras en pig latin
```

```
for word in message.split():
```

```
# Separa lo que no son letras al principio de esta palabra:
```

```
prefixNonLetters = ''
```

```
while len(word) > 0 and not word[0].isalpha():
```

```
prefixNonLetters += word[0]
```

```
word = word[1:]
```

```
if len(word) == 0:
```

```
pigLatin.append(prefixNonLetters)
```

```
continue
```

Necesitamos que cada palabra esté en su propia cadena, así que llamamos a `message.split()` para obtener una lista de las palabras como cadenas separadas. La cadena 'My name is AL SWEIGART and I am 4,000 years old.' haría que `split()` devolviese ['My', 'name', 'is', 'AL', 'SWEIGART', 'and', 'I', 'am', '4,000', 'years', 'old.'].

También tenemos que quitar lo que no sean letras del principio y el

final de cada palabra para que palabras como ‘old.’ se traduzcan como ‘oldyay.’ en vez de como ‘old.yay’. Guardamos estos caracteres que no son letras en una variable llamada `prefixNonLetters`.

Separa lo que no son letras al final de esta palabra:

```
suffixNonLetters = “
```

```
while not word[-1].isalpha():
```

```
suffixNonLetters += word[-1] + suffixNonLetters
```

```
word = word[:-1]
```

Un bucle que llama a `isalpha()` con el primer carácter de la palabra determina si deberíamos quitar un carácter de la palabra y concatenarlo al final de `prefixNonLetters`. Si toda la palabra está hecha de caracteres que no son letras, como ‘4,000’, podemos simplemente añadirla a la lista `pigLatin` y seguir con la siguiente palabra para traducir. También necesitamos guardar los caracteres que no son letras al final de la cadena de palabras. Este código es similar al bucle anterior.

A continuación, nos aseguramos de que el programa recuerde si la letra estaba en mayúsculas o empezaba por mayúscula para restaurarla después de la traducción a *pig latin*:

Recuerda si la palabra iba en mayúsculas o con mayúscula inicial:

```
wasUpper = word.isupper()
```

```
wasTitle = word.istitle()
```

```
word = word.lower() # Pon la palabra en minúsculas para la traducción.
```

Durante el resto del código del bucle `for`, trabajaremos con una versión en minúsculas de `word`.

Para convertir una palabra como “sweigart” en “eigart-sway”, hay que eliminar todas las consonantes del principio de `word`:

Separa las consonantes al principio de esta palabra:

```
prefixConsonants = ''
```

while len(word) > 0 and not word[0] in VOWELS:

```
prefixConsonants += word[0]
```

```
word = word[1:]
```

Utilizamos un bucle similar al empleado para eliminar lo que no son letras del principio de una palabra, pero ahora vamos a quitar las consonantes y a guardarlas en una variable llamada prefixConsonants.

Si había consonantes al principio de la palabra, ahora estarán en prefixConsonants y deberíamos concatenar esa variable y la cadena 'ay' al final de la palabra. En caso contrario, podemos asumir que la palabra empieza por vocal y solo tenemos que concatenar 'yay':

Añade el sufijo de pig latin a la palabra:

```
if prefixConsonants != '':
```

```
word += prefixConsonants + 'ay'
```

```
else:
```

```
word += 'yay'
```

Recuerda que pusimos word en su versión en minúscula con word = word.lower(). Si la palabra estaba originalmente en mayúsculas o empezaba con mayúscula, este código volverá a ponerla así:

Pon otra vez la palabra en mayúsculas o con mayúscula inicial:

```
if wasUpper:
```

```
word = word.upper()
```

```
if wasTitle:
```

```
word = word.title()
```

Al final del bucle `for`, añadimos la palabra, junto con cualquier prefijo o sufijo que no fuese una letra que tuviese originalmente a la lista `pigLatin`:

```
# Vuelve a poner lo que no son letras al principio o al final de la palabra.
```

```
pigLatin.append(prefixNonLetters + word + suffixNonLetters)
```

```
# Vuelve a unir todas las palabras en una única cadena:
```

```
print(' '.join(pigLatin))
```

Cuando termina este bucle, combinamos la lista de cadenas en una única cadena llamando al método `join()` y luego pasamos esta cadena a `print()` para mostrar el texto en *pig latin* en la pantalla.

Resumen

El texto es una forma de datos habitual y Python incluye muchos métodos de cadena útiles para procesar el texto guardado en valores de cadena. Usarás índices, rebanadas y métodos de cadena en casi cualquier programa de Python que escribas.

Los programas que estamos haciendo ahora no parecen muy sofisticados; no tienen interfaces gráficas de usuario (GUI) con imágenes y texto de colores. De momento, estamos mostrando texto con `print()` y dejando que el usuario escriba texto con `input()`. Sin embargo, el usuario puede introducir con rapidez grandes cantidades de texto a través del portapapeles. Esta capacidad es una vía útil para escribir programas que manipulan cantidades masivas de texto. Estos programas basados en texto no tienen ventanas ostentosas ni gráficos, pero pueden hacer mucho trabajo útil deprisa.

Otra forma de manipular grandes cantidades de texto es leer y escribir archivos directamente desde el disco duro. Veremos cómo hacer eso con Python en el capítulo 10.

Con esto hemos cubierto prácticamente todos los conceptos básicos de la programación con Python. Seguirás aprendiendo nuevos

conceptos a lo largo del libro, pero ya sabes lo suficiente para empezar a escribir programas útiles que puedan automatizar tareas. Si quieres ver una colección de programillas sencillos de Python creados a partir de las nociones básicas que ya has aprendido, puedes leer mi otro libro, *The Big Book of Small Python Projects* (No Starch Press, 2021). Prueba a copiar el código fuente de cada programa a mano y luego haz modificaciones para ver cómo afectan al comportamiento del programa. Una vez que entiendas cómo funciona el programa, intenta recrearlo desde cero. No hace falta que recries el código fuente exactamente; simplemente céntrate en lo que hace el programa más que en cómo lo hace.

Es posible que creas que no tienes suficientes conocimientos de Python para hacer cosas como descargar páginas web, actualizar hojas de cálculo o enviar mensajes de texto, pero ahí es donde entran en juego los módulos de Python. Estos módulos, escritos por otros programadores, ofrecen funciones que hacen que realizar todas esas tareas sea fácil. En el próximo capítulo, veremos cómo crear programas reales que hagan tareas útiles automatizadas.

Preguntas prácticas

1. ¿Qué son los caracteres de escape?
2. ¿Qué representan los caracteres de escape `\n` y `\t`?
3. ¿Cómo podemos poner un carácter de barra invertida (`\`) en una cadena?
4. La cadena `"Howl's Moving Castle"` es una cadena válida. ¿Por qué no es un problema que la comilla simple de `Howl's` no lleve carácter de escape?
5. Si no quieres poner `\n` en una cadena, ¿cómo puedes escribirla con saltos de línea?
6. ¿Cómo se evalúan las siguientes expresiones?
 - `'Hello, world!'`[1]
 - `'Hello, world!'`[0:5]

•'Hello, world!':[5

•'Hello, world!':[3:

7.¿Cómo se evalúan las siguientes expresiones?

•'Hello'.upper()

•'Hello'.upper().isupper()

•'Hello'.upper().lower()

8.¿Cómo se evalúan las siguientes expresiones?

•'Remember, remember, the fifth of November.'.split()

•'-'.join('There can be only one.'.split())

9.¿Qué métodos de cadena podemos usar para justificar a la derecha, justificar a la izquierda y centrar un texto?

10.¿Cómo podemos eliminar caracteres de espacio en blanco del principio o el final de una cadena?

Programa de práctica: impresión de tablas

Para practicar, escribe una función llamada `printTable()` que coja una lista de listas de cadenas y las muestre en una tabla bien organizada, con cada columna justificada a la derecha. Vamos a suponer que todas las listas internas contienen el mismo número de cadenas. Por ejemplo, el valor podría ser algo así:

```
tableData = [['apples', 'oranges', 'cherries', 'banana'],  
             ['Alice', 'Bob', 'Carol', 'David'],  
             ['dogs', 'cats', 'moose', 'goose']]
```

La función `printTable()` imprimiría lo siguiente:

```
Apples  
Bananas
```

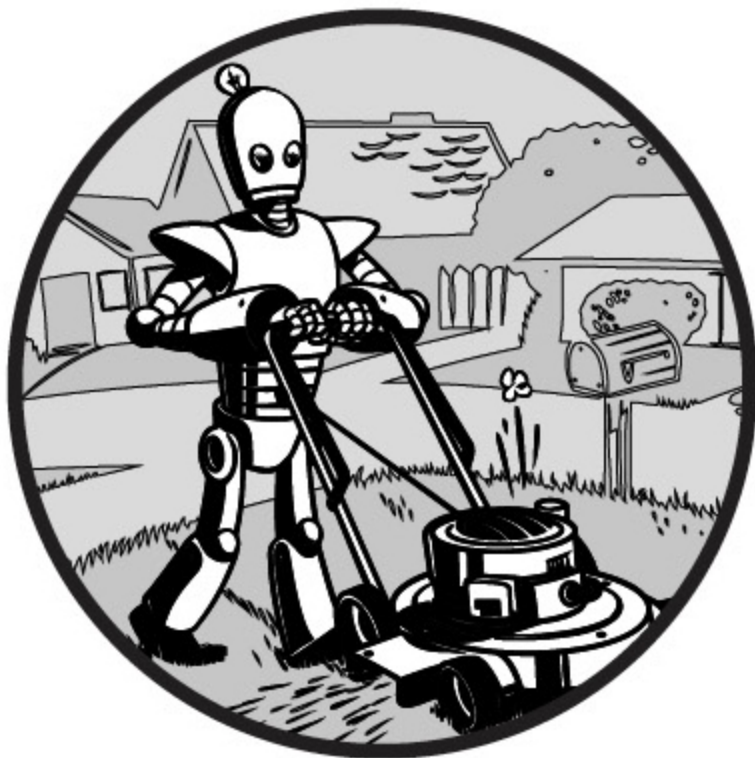
~~Charles~~
~~David~~

Pista: tu código tendrá que buscar primero la cadena más larga de cada lista para que la columna sea lo bastante ancha como para alojar todas las cadenas. Puedes almacenar la anchura máxima de cada columna como una lista de enteros. La función `printTable()` puede empezar con `colWidths = [0 * len(tableData)]`, lo cual creará una lista que contenga el mismo número de valores 0 que la cantidad de listas internas en `tableData`. De ese modo, `colWidths[0]` puede almacenar la anchura de la cadena más larga de `tableData[0]`, `colWidths[1]` puede almacenar la anchura de la cadena más larga de `tableData[1]`, y así sucesivamente. Después, puedes buscar el valor más grande de la lista `colWidths` para descubrir el entero de anchura que hay que pasar al método de cadena `rjust()`.

Parte II

Automatización de tareas

Coincidencia de patrones de texto con expresiones regulares



Seguramente estás familiarizado con el proceso de búsqueda de texto pulsando **Control-F** y escribiendo las palabras que quieres buscar. Las expresiones regulares van un paso más allá: nos permiten especificar un patrón de texto que buscar. Es posible que no recuerdes exactamente el número de teléfono de un establecimiento, pero, si vives en Estados Unidos o Canadá, sabes que constará de un prefijo de área de tres dígitos seguido de un guion, otros tres dígitos, otro guion y otros cuatro dígitos. Así es como tú, un humano, reconoces un número de teléfono cuando lo ves: 415-555-1234 es un número de teléfono, pero 4 155 551 234 \$ no lo es.

Reconocemos todo tipo de patrones de texto cada día: las direcciones de correo electrónico tienen símbolos @ en medio, los números de la Seguridad Social de Estados Unidos tienen nueve dígitos y dos guiones, las URL de sitios web suelen tener puntos y barras inclinadas, el DNI español son ocho números y una letra y los *hashtags* de las redes sociales empiezan por # y no contienen espacios, por poner algunos ejemplos.

Las expresiones regulares son útiles, pero muy poca gente ajena a la programación conoce su existencia, aunque la mayoría de los editores y procesadores de texto moderno tienen funciones de búsqueda y reemplazo que pueden buscar basándose en expresiones regulares. Estas expresiones ahorran muchísimo tiempo, no solo a los usuarios de software, sino también a los programadores. De hecho, en el artículo de *The Guardian* “Here’s What ICT Should Really Teach Kids: How to Do Regular Expressions”, el experto en tecnología Cory Doctorow argumenta que deberíamos enseñar expresiones regulares antes de enseñar a programar:

Conocer [expresiones regulares] puede suponer la diferencia entre resolver un problema en 3 pasos o hacerlo en 3.000 pasos. Cuando eres un cerebritito, olvidas que los problemas que tú resuelves tocando un par de teclas pueden llevar a otras personas un esfuerzo de días de trabajo tedioso y propenso a errores.

En este capítulo, empezaremos escribiendo un programa que busque patrones de texto sin usar expresiones regulares y luego aprenderás a usar expresiones regulares para simplificar el código. Te mostraré la coincidencia básica con expresiones regulares y luego avanzaremos a funciones más potentes, como la sustitución de cadenas y la creación de tus propias clases de caracteres. También aprenderás a usar el módulo *Humre*, que ofrece sustitutos en inglés normal para la críptica sintaxis basada en símbolos de las expresiones regulares.

Buscar patrones de texto sin expresiones regulares

Imagina que quieres buscar un número de teléfono estadounidense en una cadena; estás buscando tres números, un guion, tres números, un guion y cuatro números. Aquí tienes un ejemplo:

415-555-4242.

Vamos a escribir una función llamada `is_phone_number()` para que compruebe si una cadena coincide con este patrón y devuelva `True` o `False`. Abre una pestaña nueva en el editor de archivos, escribe este código y guárdalo como `isPhoneNumber.py`:

```
def is_phone_number(text):
```

```
    1 if len(text) != 12: # Los números de teléfono tienen exactamente  
    12 caracteres.
```

```
    return False
```

```
    for i in range(0, 3): # Los primeros tres caracteres deben ser  
    números.
```

```
    2 if not text[i].isdecimal():
```

```
    return False
```

```
    3 if text[3] != '-': # El cuarto carácter debe ser un guion.
```

```
    return False
```

```
    for i in range(4, 7): # Los tres caracteres siguientes deben ser  
    números.
```

```
    4 if not text[i].isdecimal():
```

```
    return False
```

```
    5 if text[7] != '-': # El octavo carácter debe ser un guion.
```

```
    return False
```

```
    for i in range(8, 12): # Los tres caracteres siguientes deben ser  
    números.
```

```
    6 if not text[i].isdecimal():
```

```
    return False
```

7 return True

```
print('Is 415-555-4242 a phone number?',  
is_phone_number('415-555-4242'))
```

```
print(is_phone_number('415-555-4242'))
```

```
print('Is Moshi moshi a phone number?', is_phone_number('Moshi  
moshi'))
```

```
print(is_phone_number('Moshi moshi'))
```

Cuando ejecutes este programa, la salida debería ser como esto:

Is 415-555-4242 a phone number?

True

Is Moshi moshi a phone number?

False

La función `is_phone_number()` tiene código que hace varias comprobaciones para determinar si la cadena en `text` es un número de teléfono válido. Si alguna de estas comprobaciones falla, la función devuelve `False`. Primero, el código comprueba que la cadena tenga exactamente 12 caracteres 1. A continuación, comprueba que el código de área (esto es, los primeros tres caracteres) son solo caracteres numéricos 2 llamando al método de cadena `isdecimal()`. El resto de la función comprueba si la cadena sigue el patrón de un número de teléfono: el número debe tener el primer guion después del prefijo de área 3, tres caracteres numéricos 4, otro guion 5 y, por último, otros cuatro caracteres numéricos 6. Si la ejecución del programa supera todas las comprobaciones, devuelve `True` 7.

Llamar a `is_phone_number()` con el argumento `'415-555-4242'` devolverá `True`. Llamar a `is_phone_number()` con `'Moshi moshi'` devolverá `False`; la primera prueba ya falla porque `'Moshi moshi'` no tiene 12 caracteres.

Si quisiéremos encontrar un número de teléfono en una cadena más

grande, habría que añadir más código todavía para localizar el patrón. Sustituye las cuatro últimas llamadas a `print()` de `isPhoneNumber.py` por lo siguiente:

```
message = 'Call me at 415-555-1011 tomorrow. 415-555-9999 is my office.'
```

```
for i in range(len(message)):
```

```
    1 segment = message[i:i + 12]
```

```
    2 if is_phone_number(segment):
```

```
        print('Phone number found: ' + segment)
```

```
    print('Done')
```

Al ejecutar el programa, la salida será así:

```
Phone number found: 415-555-1011
```

```
Phone number found: 415-555-9999
```

```
Done
```

En cada iteración del bucle `for`, se asigna un nuevo segmento de 12 caracteres de `message` a la variable `segment` 1. Por ejemplo, en la primera iteración, `i` es 0 y se asigna a `segment` `message[0:12]` (esto es, la cadena 'Call me at 4'). En la siguiente iteración, `i` es 1 y se asigna a `segment` `message[1:13]` (la cadena 'all me at 41'). En otras palabras, en cada iteración del bucle `for`, `segment` toma los siguientes valores:

```
'Call me at 4'
```

```
'all me at 41'
```

```
'll me at 415'
```

```
'l me at 415-'
```

Y así sucesivamente hasta el último valor, que es 's my office.'.

El código del bucle pasa `segment` a `is_phone_number()` para comprobar si coincide con el patrón de número de teléfono 2 y, si es así, imprime el segmento. Una vez que ha terminado de pasar por el mensaje, imprime `Done`.

Aunque, en este ejemplo, la cadena de `message` es corta, el programa se ejecutaría en menos de un segundo incluso aunque tuviese millones de caracteres. Un programa similar que busque números de teléfono usando expresiones regulares también se ejecutaría en menos de un segundo; no obstante, las expresiones regulares hacen que sea mucho más rápido escribir este programa.

Buscar patrones de texto con expresiones regulares

El programa anterior para buscar un número de teléfono funciona, pero usa mucho código para hacer algo limitado. La función `is_phone_number()` tiene 17 líneas, pero solo puede buscar un formato de número de teléfono. ¿Qué pasaría si el número tuviese un formato diferente, como 415.555.4242 o (415) 555-4242? ¿Y si el número llevase una extensión, como 415-555-4242 x99? La función `is_phone_number()` no los encontraría. Podríamos añadir más código para esos patrones adicionales, pero hay una manera más sencilla de resolver el problema.

Las expresiones regulares, llamadas *regex* para abreviar, son una especie de minilenguaje que describe un patrón de texto. Por ejemplo, los caracteres `\d` en una expresión regular representan un número decimal entre 0 y 9. Python usa la cadena `r'\d\d\d\d\d\d\d\d\d\d'` para buscar el mismo patrón de texto que la función `is_phone_number()`: una cadena de tres números, un guion, tres números más, otro guion y otros cuatro números. Cualquier otra cadena no coincidiría con la expresión regular `r'\d\d\d\d\d\d\d\d\d\d'`.

Las expresiones regulares pueden ser mucho más sofisticadas que esta. Por ejemplo, añadir un numeral, como 3, entre llaves (`{3}`) después de un patrón es como decir: “Busca este patrón tres veces”. Así que la expresión regular ligeramente más corta `r'\d{3}-\d{3}-\d{4}'` también coincide con el número de teléfono.

Observa que a menudo escribimos cadenas de expresiones regulares como cadenas crudas, con el prefijo `r`. Esto es útil, ya que las cadenas de expresiones regulares a menudo contienen barras invertidas. Sin usar cadenas crudas, tendríamos que introducir expresiones como `'\\d'`.

Antes de ver todos los detalles de la sintaxis de las expresiones regulares, vamos a hablar de cómo se usan en Python. Seguiremos con el ejemplo de la cadena `r'\d{3}-\d{3}-\d{4}'` utilizada para encontrar números de teléfono estadounidenses en una cadena de texto `'My number is 415-555-4242'`. El proceso general para usar expresiones regulares en Python implica cuatro pasos:

1. Importar el módulo `re`.
2. Pasar la cadena de expresión regular a `re.compile()` para obtener un objeto `Pattern`.
3. Pasar la cadena de texto al método `search()` del objeto `Pattern` para obtener un objeto `Match`.
4. Llamar al método `group()` del objeto `Match` para obtener la cadena del texto coincidente.

En el intérprete de comandos interactivo, estos pasos quedan así:

```
> > > import re

> > > phone_num_pattern_obj = re.compile(r'\d{3}-\d{3}-\d{4}')

> > > match_obj = phone_num_pattern_obj.search('My
number is 415-555-4242.')

> > > match_obj.group()

'415-555-4242'
```

Todas las funciones de expresiones regulares de Python están en el módulo `re`. La mayoría de los ejemplos de este capítulo requerirán este módulo, así que acuérdate de importarlo al principio del programa. Si no lo haces, te saldrá un mensaje de error `NameError`:

name 're' is not defined. Como al importar cualquier módulo, solo hace falta importarlo una vez por programa o por sesión del intérprete de comandos interactivo. Pasar la cadena de expresión regular a `re.compile()` devuelve un objeto `Pattern`. Solo necesitamos compilar el objeto `Pattern` una vez; después, podemos llamar a su método `search()` para tantas cadenas de texto diferentes como queramos.

El método `search()` del objeto `Pattern` busca la cadena que se le pasa en busca de cualquier coincidencia con la expresión regular. El método `search()` devolverá `None` si no encuentra el patrón de la expresión regular en la cadena. Si encuentra el patrón, devuelve un objeto `Match`, que tendrá un método `group()` que devuelve una cadena del texto coincidente.

Nota *Te animo a introducir el código de ejemplo en el intérprete de comandos interactivo, también podrías usar probadores de expresiones regulares basados en web, que te muestran exactamente como una regex coincide con un fragmento de texto que introduces. Yo recomiendo los probadores de <https://pythex.org> y <https://regex101.com>. Los distintos lenguajes de programación pueden tener una sintaxis de expresiones regulares ligeramente diferente, así que asegúrate de elegir Python en estos sitios web.*

La sintaxis de las expresiones regulares

Ahora que conoces los pasos básicos para crear y localizar objetos de expresiones regulares con Python, estás listo para aprender la gama completa de sintaxis de expresiones regulares. En este apartado, veremos cómo agrupar elementos de expresiones regulares con paréntesis, cómo usar caracteres de escape especiales, cómo unir varios grupos alternativos con el carácter barra vertical y cómo devolver todas las coincidencias con el método `findall()`.

Agrupar con paréntesis

Supongamos que queremos separar una parte más pequeña del texto coincidente, como el prefijo de área, del resto del número de teléfono (para, por ejemplo, hacer alguna operación con él). Añadir paréntesis creará grupos en la cadena de expresión regular: `r'(\d\d`

`\d)-(\d\d\d-\d\d\d\d)`'. Después, podemos usar el método `group()` de los objetos `Match` para coger el texto coincidente de un solo grupo.

El primer par de paréntesis de una cadena de expresiones regulares será el grupo 1, el segundo, el 2. Pasando el entero 1 o 2 al método `group()`, podemos coger distintas partes del texto coincidente. Pasar 0 o nada a este método devolverá todo el texto coincidente. Escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > import re
```

```
> > > phone_re = re.compile(r'(\d\d\d)-(\d\d\d-\d\d\d\d)')
```

```
> > > mo = phone_re.search('My number is 415-555-4242.')
```

```
> > > mo.group(1) # Devuelve el primer grupo de texto coincidente.
```

```
'415'
```

```
> > > mo.group(2) # Devuelve el segundo grupo de texto coincidente.
```

```
'555-4242'
```

```
> > > mo.group(0) # Devuelve todo el texto coincidente.
```

```
'415-555-4242'
```

```
> > > mo.group() # También devuelve todo el texto coincidente.
```

```
'415-555-4242'
```

Si quisiéramos recuperar todos los grupos a la vez, utilizaríamos el método `groups()` (fíjate en que el nombre está en plural):

```
> > mo.groups()
```

```
('415', '555-4242')
```

```
> > > area_code, main_number = mo.groups()
```

```
> > > print(area_code)
```

```
415
```

```
> > > print(main_number)
```

```
555-4242
```

Como `mo.groups()` devuelve una tupla de múltiples valores, podemos usar el truco de asignación múltiple para asignar cada valor a una variable aparte, como en la línea anterior `area_code`, `main_number = mo.groups()`.

Usar caracteres de escape

Los paréntesis crean grupos en expresiones regulares y no se interpretan como parte del patrón de texto. Entonces, ¿qué hacemos si tenemos que buscar la coincidencia de paréntesis en un texto? Por ejemplo, a lo mejor el número de teléfono tiene el prefijo entre paréntesis: ‘(415) 555-4242’.

En este caso, hay que acompañar los caracteres (y) con una barra invertida. Los caracteres de escape `\(` y `\)` se interpretarán como parte del patrón que estamos buscando. Escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > pattern = re.compile(r'(\d\d\d)) (\d\d\d-\d\d\d\d)')
```

```
> > > mo = pattern.search('My phone number is (415) 555-4242.')
```

```
> > > mo.group(1)
```

```
‘(415)’
```

```
> > > mo.group(2)
```

```
‘555-4242’
```

Los caracteres de escape `\(` y `\)` en la cadena cruda pasada a `re.compile()` coincidirán con los paréntesis reales. En las expresiones regulares, los siguientes caracteres tienen significados especiales:

`$ ( ) * + - . ? [ \ ] ^ { | }`

Si queremos detectar estos caracteres como parte de nuestro patrón de texto, hay que usar la barra invertida como carácter de escape:

`\$ \( \) \* \+ \- \. \? \[ \] \^ \{ \| \}`

Comprueba siempre que no has confundido los paréntesis `\( y \)` con los paréntesis normales `( y )` en una expresión regular. Si recibes un mensaje de error sobre missing `)` o unbalanced parenthesis, es posible que hayas olvidado el paréntesis de cierre normal de un grupo, como en este ejemplo:

```
> > > import re
> > > re.compile(r'\(Parentheses\)')
```

Traceback (most recent call last):

--fragmento omitido--

re.error: missing `)`, unterminated subpattern at position 0

El mensaje de error nos indica que hay un paréntesis de apertura en el índice 0 de la cadena `r'\(Parentheses\)`' y falta el correspondiente paréntesis de cierre. El uso del módulo `Humre` descrito más adelante en este capítulo ayuda a evitar este tipo de erratas.

Coincidencia de caracteres de grupos alternos

El carácter `|` se llama barra vertical y se usa como operador de alternancia en las expresiones regulares. Podemos usarlo en cualquier parte donde queramos buscar una de varias expresiones. Por ejemplo, la expresión regular `r'Cat|Dog'` coincidirá con `'Cat'` o con `'Dog'`.

También se puede usar la barra vertical para buscar uno de varios patrones como parte de una expresión regular. Por ejemplo, supongamos que queremos encontrar cualquiera de estas cadenas `'Caterpillar'`, `'Catastrophe'`, `'Catch'` o `'Category'`. Como todas

empiezan por Cat, estaría bien poder especificar ese prefijo solo una vez. Podemos hacerlo con la barra vertical entre paréntesis para separar los posibles sufijos. Escribe esto en el intérprete de comandos interactivo:

```
> > > import re
> > > pattern = re.compile(r'Cat(erpillar|astrophe|ch|egory)')
> > > match = pattern.search('Catch me if you can.')
> > > match.group()
'Catch'
> > > match.group(1)
'ch'
```

La llamada al método `match.group()` devuelve todo el texto que coincide plenamente 'Catch', mientras que `match.group(1)` devuelve solo la parte del texto coincidente dentro del primer grupo de paréntesis, 'ch'. Con la barra vertical y los paréntesis para agrupar, podemos especificar varios patrones alternativos con los que nos gustaría que coincidiese nuestra expresión regular. Si necesitas buscar una barra vertical, usa la barra invertida como carácter de escape, `\|`.

Devolver todas las coincidencias

Además de un método `search()`, los objetos `Pattern` tienen un método `findall()`. Mientras que `search()` devuelve un objeto `Match` para el primer texto coincidente en la cadena buscada, `findall()` devuelve las cadenas de todas las coincidencias en la cadena buscada. Hay un detalle que debes recordar cuando uses `findall()`. El método devuelve una lista de cadenas siempre y cuando no haya grupos en la expresión regular. Escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > import re
```

```
> > > pattern = re.compile(r'\d{3}-\d{3}-\d{4}') # This regex  
has no groups.
```

```
> > > pattern.findall('Cell: 415-555-9999 Work:  
212-555-0000')
```

```
['415-555-9999', '212-555-0000']
```

Si hay grupos en la expresión regular, `findall()` devuelve una lista de tuplas. Cada tupla representa una sola coincidencia y tiene cadenas para cada grupo de la expresión regular. Para ver este comportamiento en acción, escribe lo siguiente en el intérprete de comandos interactivo (y observa que la expresión regular compilada ahora tiene grupos entre paréntesis):

```
> > > import re
```

```
> > > pattern = re.compile(r'(\d{3})-(\d{3})-(\d{4}')
```

```
> > > pattern.findall('Cell: 415-555-9999 Work:  
212-555-0000')
```

```
[('415', '555', '9999'), ('212', '555', '0000')]
```

Ten en cuenta también que `findall()` no solapa coincidencias. Por ejemplo, la coincidencia de tres números con la cadena `r'\d{3}'` usa los tres primeros números de `'1234'` pero no los tres últimos:

```
> > > import re
```

```
> > > pattern = re.compile(r'\d{3}')
```

```
> > > pattern.findall('1234')
```

```
['123']
```

```
> > > pattern.findall('12345')
```

```
['123']
```

```
> > > pattern.findall('123456')
```

[‘123’, ‘456’]

Como los tres primeros dígitos de ‘1234’ han coincidido como ‘123’, los dígitos ‘234’ no se incluirán en otras coincidencias aunque encajen en el patrón `r\d{3}`.

Sintaxis calificadora: qué caracteres hacer coincidir

Las expresiones regulares se dividen en dos partes: calificadores, que dictan qué caracteres intentamos hacer coincidir, y cuantificadores, que determinan cuántos caracteres intentamos hacer coincidir. En el ejemplo de la cadena de expresión regular de teléfono `r\d{3}-\d{3}-\d{4}` que hemos estado usando, las partes `r\d` y `-` son calificadores y las partes `{3}` y `{4}` son cuantificadores. Vamos a analizar la sintaxis de los calificadores.

Usar clases de caracteres y clases de caracteres negativas

Aunque podemos definir un único carácter para la coincidencia, como hemos hecho en los ejemplos anteriores, también podemos definir un conjunto de caracteres si los ponemos entre corchetes. Este conjunto se denomina “clase de caracteres”. Por ejemplo, la clase de caracteres `[aeiouAEIOU]` coincidirá con cualquier vocal, ya sea mayúscula o minúscula. Equivale a poner `a|e|i|o|u|A|E|I|O|U`, pero es más fácil de escribir. Introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > import re
```

```
> > > vowel_pattern = re.compile(r'[aeiouAEIOU]')
```

```
> > > vowel_pattern.findall('RoboCop eats BABY FOOD.')
```

[‘o’, ‘o’, ‘o’, ‘e’, ‘a’, ‘A’, ‘O’, ‘O’]

También podemos incluir rangos de letras o números utilizando un guion. Por ejemplo, la clase de caracteres `[a-zA-Z0-9]` coincidirá con todas las letras, mayúsculas o minúsculas, y números. Observa que, dentro de los corchetes, los símbolos de expresión regular normales no se interpretan como tales. Esto significa que no es necesario usar caracteres de escape para buscar paréntesis reales dentro de los

corchetes. Por ejemplo, la clase de caracteres `[]` buscará un paréntesis de apertura o de cierre, no hace falta escribir esto como `[\(\)].`

Colocando un signo de intercalación (`^`) justo después del corchete de apertura de una clase de caracteres la convierte en una clase negativa. Una clase de caracteres negativa buscará todos los caracteres que no estén en la clase de caracteres. Por ejemplo, escribe esto en el intérprete de comandos interactivo:

```
> > > import re
```

```
> > > consonant_pattern = re.compile(r'[^aeiouAEIOU]')
```

```
> > > consonant_pattern.findall('RoboCop eats BABY FOOD.')
```

```
['R', 'b', 'C', 'p', ' ', 't', 's', ' ', 'B', 'B', 'Y', ' ', 'F', 'D', '.']
```

Ahora, en lugar de buscar todas las vocales, estamos haciendo coincidir todos los caracteres que no sean una vocal. Ten en cuenta que esto incluye espacios, saltos de línea, caracteres de puntuación y números.

Utilizar clases de caracteres estenográficas

En el anterior ejemplo de la expresión regular del número de teléfono, vimos que `\d` puede ser cualquier dígito numérico. Es decir, `\d` es la estenografía de la expresión regular `0|1|2|3|4|5|6|7|8|9` o `[0-9]`. Hay muchos atajos de este tipo, como se muestra en la tabla 9.1.

Tabla 9.1. Códigos estenográficos para clases de caracteres habituales.

Representación de caracteres estenográfica

Cualquier dígito numérico de 0 a 9.

Cualquier carácter que no sea un dígito numérico de 0 a 9.

Cualquier letra, dígito numérico o guion bajo. (Piensa en esto como caracteres de "palabra").

Cualquier carácter que no sea una letra, dígito numérico o guion bajo.

Cualquier espacio, tabulación o salto de línea. (Piensa en esto como caracteres de "espacio").

Cualquier carácter que no sea un espacio, tabulación o salto de línea.

Observa que, aunque `\d` hace coincidir dígitos y `\w` dígitos, letras y el guion bajo, no hay una clase de caracteres estenográfica que busque solo letras. Aunque podamos usar la clase de caracteres `[a-zA-Z]`, esta no buscará letras con acento o letras de un alfabeto distinto del romano, como 'é'. Además, hay que usar cadenas crudas para usar la barra invertida como carácter de escape: `r'\d'`.

Por ejemplo, escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > import re
```

```
> > > pattern = re.compile(r'\d + \s\w +')
```

```
> > > pattern.findall('12 drummers, 11 pipers, 10 lords, 9  
ladies, 8 maids,
```

```
7 swans, 6 geese, 5 rings, 4 birds, 3 hens, 2 doves, 1 partridge')
```

```
['12 drummers', '11 pipers', '10 lords', '9 ladies', '8 maids', '7  
swans', '
```

```
6 geese', '5 rings', '4 birds', '3 hens', '2 doves', '1 partridge']
```

La expresión regular `\d + \s\w +` buscará cualquier texto que tenga uno o más dígitos numéricos (`\d +`), seguido de un carácter de espacio en blanco (`\s`), seguido de uno o varios caracteres de letras/dígitos/guiones bajos (`\w +`). El método `findall()` devuelve todas las cadenas coincidentes con el patrón de la expresión regular en una lista.

Hacer coincidir todo con el carácter punto

El carácter `.` (o punto) en la cadena de una expresión regular busca cualquier carácter excepto un salto de línea. Por ejemplo, escribe esto en el intérprete de comandos interactivo:

```
> > > import re
```

```
> > > at_re = re.compile(r'.at')
```

```
> > > at_re.findall('The cat in the hat sat on the flat mat.')
```

```
['cat', 'hat', 'sat', 'lat', 'mat']
```

Recuerda que el punto buscará solo un carácter, por eso, en el texto flat del ejemplo anterior solo coincide lat. Para buscar un punto de verdad, hay que ponerlo con la barra invertida: \.

Ten cuidado con lo que haces coincidir

Lo mejor y lo peor de las expresiones regulares es que coinciden exactamente con lo que pides. Hay algunos puntos de confusión en lo que respecta a las clases regulares:

- La clase de caracteres [A-Z o [a-z busca letras en mayúscula o en minúscula, respectivamente, pero no ambas. Para eso, hay que usar [A-Za-z.

- La clase de caracteres [A-Za-z solo busca caracteres simples y sin acento. Por ejemplo, la cadena de expresión regular r'First Name: ([A-Za-z] +)' coincidiría con First Name: seguido de una o varias letras sin acento. Pero el nombre de la cantante Sinéad O'Connor solo llegaría hasta la é y el grupo se establecería en 'Sin'.

- La clase de caracteres \w busca todas las letras, incluidas las letras acentuadas y caracteres de otros alfabetos. Pero también números y guiones bajos, así que la cadena de expresiones regulares r'First Name: (\w +)' puede coincidir más de lo que pretendíamos.

- La clase de caracteres \w busca todas las letras, pero la cadena r'Last Name: (\w +)' cogería el apellido de Sinéad O'Connor solo hasta el apóstrofo. Esto significa que el grupo lo tomaría como 'O'.

- Los caracteres de comillas rectas e inglesas (" ' " ') se consideran totalmente diferentes entre sí, por lo que deben especificarse por separado.

Los datos del mundo real son complicados. Incluso aunque el programa consiga coger bien el nombre de Sinéad O'Connor, fracasaría con el de Jean-Paul Sartre por el guion.

Naturalmente, cuando el software declare un nombre como entrada no válida, es el software y no el nombre lo que falla; los nombres de las personas no pueden ser no válidos. Puedes descubrir más sobre este asunto con el artículo de Patrick McKenzie “Falsehoods Programmers Believe About Names”, en <https://www.kalzumeus.com/2010/06/17/falsehoods-programmers-believe-about-names/>. Este artículo dio lugar a una serie de artículos similares sobre “falsedades que creen los programadores” acerca de cómo el software maneja incorrectamente las fechas, las zonas horarias, las divisas, las direcciones postales, los géneros, los códigos de aeropuerto y el amor. Puedes ver la charla de Carina C. Zona en PyCon 2015 sobre este tema, “Schemas for the Real World”, en <https://youtu.be/PYYfVqtcWQY>.

Sintaxis de cuantificadores: cuántos calificadores buscar

En una cadena de expresión regular, los cuantificadores siguen a los calificadores para determinar cuántos de ellos buscar. Por ejemplo, en la expresión regular del número de teléfono de antes, {3} sigue a \d para buscar exactamente tres dígitos. Si no hay cuantificador detrás del calificador, el calificador debe aparecer solo una vez: puedes pensar que r'\d' es lo mismo que r'\d{1}'.

Buscar un patrón opcional

A veces, puede interesarnos buscar un patrón solo opcionalmente. Es decir, la experiencia regular debería buscar cero o uno de los calificadores anteriores. El carácter ? marca el calificador precedente como opcional. Para ver un ejemplo, escribe esto en el intérprete de comandos interactivo:

```
> > > import re
> > > pattern = re.compile(r'42!?')
> > > pattern.search('42!')
```



```
<re.Match object; span = (0, 3), match = '42! '>
```

```
> > > pattern.search('42')
```

```
<re.Match object; span = (0, 2), match = '42'>
```

La parte ? de la expresión regular significa que el patrón ! es opcional. Por eso coinciden tanto 42! (con la exclamación) y 42 (sin ella).

Como verás, la dependencia de la sintaxis de las expresiones regulares de símbolos y puntuaciones la hace difícil de leer: el interrogante ? tiene significado en la sintaxis de las expresiones regulares, pero la exclamación ! no. Así pues, r'42!?' significa '42' seguido opcionalmente por una exclamación, pero r'42?!' significa '4' seguido opcionalmente de '2' seguido de '!':

```
> > > import re
```

```
> > > pattern = re.compile(r'42?!')
```

```
> > > pattern.search('42!')
```

```
<re.Match object; span = (0, 3), match = '42! '>
```

```
> > > pattern.search('4!')
```

```
<re.Match object; span = (0, 2), match = '4! '>
```

```
> > > pattern.search('42') = None # No hay coincidencia
```

True

Para hacer varios caracteres opcionales, mételos en un grupo y pon el ? después del grupo. En el ejemplo anterior del número de teléfono, podemos usar ? para hacer que la expresión regular busque números de teléfono que tengan prefijo o no. Escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > pattern = re.compile(r'(\d{3}-)?\d{3}-\d{4}')
```

```
> > > match1 = pattern.search('My number is 415-555-4242')
```

```
> > > match1.group()
```

```
'415-555-4242'
```

```
> > > match2 = pattern.search('My number is 555-4242')
```

```
> > > match2.group()
```

```
'555-4242'
```

Imagina que el interrogante es como decir: “Busca cero o uno del grupo que precede a este interrogante”.

Si necesitas buscar un interrogante de verdad, usa el carácter de escape \?.

Buscar cero o varios calificadores

El asterisco (*) significa “busca cero o más”. En otras palabras, el calificador que precede al asterisco puede aparecer cualquier número de veces en el texto. Puede estar totalmente ausente o repetirse una y otra vez. Observa el siguiente ejemplo:

```
> > > import re
```

```
> > > pattern = re.compile('Eggs( and spam)*')
```

```
> > > pattern.search('Eggs')
```

```
<re.Match object; span = (0, 4), match = 'Eggs'>
```

```
> > > pattern.search('Eggs and spam')
```

```
<re.Match object; span = (0, 13), match = 'Eggs and spam'>
```

```
> > > pattern.search('Eggs and spam and spam')
```

```
<re.Match object; span = (0, 22), match = 'Eggs and spam and spam'>
```

```
> > > pattern.search('Eggs and spam and spam and spam')
```

```
<re.Match object; span = (0, 31), match = 'Eggs and spam and spam and spam'>
```

Aunque la parte 'Eggs' de la cadena debe aparecer una vez, puede haber cualquier cantidad de ' and spam' detrás, incluida ninguna aparición.

Si hay que buscar un asterisco real, se puede prefijar el asterisco de la expresión regular con una barra invertida, *.

Buscar uno o más calificadores

Mientras que * significa “busca cero o más”, el signo más (+) significa “busca uno o más”. A diferencia del asterisco, que no requiere que su calificador aparezca en la cadena coincidente, + requiere que el calificador aparezca al menos una vez. No es opcional. Escribe lo siguiente en el intérprete de comandos interactivo y compáralo con las expresiones regulares con asterisco del apartado anterior:

```
> > > pattern = re.compile('Eggs( and spam) +')
```

```
> > > pattern.search('Eggs and spam')
```

```
<re.Match object; span = (0, 13), match = 'Eggs and spam'>
```

```
> > > pattern.search('Eggs and spam and spam')
```

```
<re.Match object; span = (0, 22), match = 'Eggs and spam and spam'>
```

```
> > > pattern.search('Eggs and spam and spam and spam')
```

```
<re.Match object; span = (0, 31), match = 'Eggs and spam and spam and spam'>
```

La expresión regular 'Eggs(and spam) +' no coincidirá con la cadena 'Eggs' porque + requiere al menos un ' and spam'.

Con frecuencia, usaremos paréntesis en las cadenas de expresiones regulares para aplicar calificadores de modo que pueda aplicarse un cuantificador a todo el grupo. Por ejemplo, podríamos buscar

cualquier combinación de puntos y guiones en código morse con `r'(\.|\\-)+'` (aunque esta expresión también devolvería combinaciones de código que no son válidas en morse). Si necesitas buscar un signo de adición de verdad, prefíjalo en la expresión regular con una barra invertida: `\\+`.

Buscar un número específico de calificadores

Si tienes un grupo que quieres repetir un determinado número de veces, pon detrás de ese grupo en la expresión regular un número entre llaves. Por ejemplo, la expresión regular `(Ha){3}` coincidirá con la cadena `'HaHaHa'` pero no con `'HaHa'`, ya que la última solo tiene dos veces el grupo `(Ha)`. En lugar de un número, se puede especificar un rango escribiendo el mínimo, una coma y el máximo entre las llaves. Por ejemplo, la expresión `(Ha){3,5}` coincidirá con `'HaHaHa'`, `'HaHaHaHa'` y `'HaHaHaHaHaHa'`.

También podemos dejar fuera el primer o el segundo número de las llaves para no limitar el mínimo o el máximo. Por ejemplo, `(Ha){3,}` coincidirá con tres o más apariciones del grupo `(Ha)`, mientras que `(Ha){,5}` coincidirá con entre cero y cinco apariciones. Las llaves pueden ayudarte a acortar tus expresiones regulares. Estas dos expresiones buscan patrones idénticos:

`(Ha){3}`

`HaHaHa`

Al igual que estas dos:

`(Ha){3,5}`

`(HaHaHa)|(HaHaHaHa)|(HaHaHaHaHaHa)`

Escribe esto en el intérprete de comandos interactivo:

```
> > > import re
```

```
> > > haRegex = re.compile(r'(Ha){3}')
```

```
> > > match1 = haRegex.search('HaHaHa')
```

```
> > > match1.group()
```

```
'HaHaHa'
```

```
> > > match = haRegex.search('HaHa')
```

```
> > > match = None
```

```
True
```

Aquí, `(Ha){3}` coincide con `'HaHaHa'` pero no con `'Ha'`. Como no coincide con `'HaHa'`, `search()` devuelve `None`.

La sintaxis del calificador con las llaves es similar a la sintaxis de rebanadas de Python (como `'Hello, world!'[3:5]`, que se evalúa a `'lo'`). Pero hay diferencias sustanciales. En el cuantificador de expresiones regulares, los dos números están separados por una coma y no por dos puntos. Además, el segundo número del cuantificador es inclusivo: `'(Ha){3,5}'` busca hasta cinco (incluido) apariciones del calificador `'(Ha)'`.

Búsquedas voraces y no voraces

Dado que `(Ha){3,5}` puede encontrar tres, cuatro o cinco apariciones de `Ha` en la cadena `'HaHaHaHaHa'`, es posible que te estés preguntando por qué la llamada del objeto `Match` a `group()` en el ejemplo anterior de las llaves devuelva `'HaHaHaHaHa'` en vez de las posibilidades más cortas. Después de todo, `'HaHaHa'` y `'HaHaHaHa'` también son coincidencias válidas para la expresión `(Ha){3,5}`.

Las expresiones regulares de Python son “voraces” por defecto, lo que significa que en una situación ambigua buscarán la cadena más larga posible. La versión no voraz (o perezosa) de las llaves, que buscará la cadena más corta posible, debe llevar un interrogante detrás de la llave de cierre.

Escribe lo siguiente en el intérprete de comandos interactivo y fíjate en la diferencia entre las versiones voraz y no voraz de la búsqueda de la misma cadena:

```
> > > import re
> > > greedy_pattern = re.compile(r'(Ha){3,5}')
> > > match1 = greedy_pattern.search('HaHaHaHaHa')
> > > match1.group()
'HaHaHaHaHa'
> > > lazy_pattern = re.compile(r'(Ha){3,5}??')
> > > match2 = lazy_pattern.search('HaHaHaHaHa')
> > > match2.group()
'HaHaHa'
```

Observa que el interrogante puede tener dos significados en las expresiones regulares: declarar una búsqueda perezosa o declarar un calificador opcional. Estos significados no tienen absolutamente nada que ver entre sí. Cabe destacar que, técnicamente, se puede pasar sin usar el cuantificador opcional `?`, o incluso los cuantificadores `*` y `+`:

- El cuantificador `?` es lo mismo que `{0,1}`.
- El cuantificador `*` es lo mismo que `{0,}`.
- El cuantificador `+` es lo mismo que `{1,}`.

Sin embargo, estos cuantificadores son un atajo habitual.

Buscar todo

A veces, puede que queramos buscar todo y cualquier cosa. Por ejemplo, supongamos que queremos buscar la cadena `'First Name:'`, seguida de cualquier texto, seguido de `'Last Name:'` y otra vez cualquier texto. Podemos usar el punto asterisco (`.``*`) para representar esa “cualquier cosa”. Recuerda que el punto significa “cualquier carácter menos el salto de línea” y el asterisco significa “cero o más del carácter anterior”. Escribe lo siguiente en el

intérprete de comandos interactivo:

```
> > > import re

> > > name_pattern = re.compile(r'First Name: ) Last Name: )

> > > name_match = name_pattern.search('First Name: Al Last
Name: Sweigart')

> > > name_match.group(1)

'Al'

> > > name_match.group(2)

'Sweigart'
```

El punto asterisco utiliza el modo voraz: siempre intenta buscar tanto texto como sea posible. Para buscar cualquier y todo el texto al estilo no voraz o perezoso, utiliza el punto asterisco y un interrogante (.?). Como cuando se utiliza con las llaves, el interrogante dice a Python que no haga una búsqueda voraz.

Escribe lo siguiente en el intérprete de comandos interactivo para ver la diferencia entre expresiones voraces y no voraces:

```
> > > import re

> > > lazy_pattern = re.compile(r'<.*?>')

> > > match1 = lazy_pattern.search('<To serve man> for
dinner.>')

> > > match1.group()

'<To serve man>'

> > > greedy_re = re.compile(r'<.*>')

> > > match2 = greedy_re.search('<To serve man> for
dinner.>')

> > > match2.group()
```

`'<To serve man> for dinner.>'`

Ambas expresiones regulares se traducen como “busca un paréntesis angular de apertura, seguido de cualquier cosa, seguida de un paréntesis angular de cierre”, pero la cadena `'<To serve man> for dinner.>'` tiene dos posibles coincidencias para el paréntesis angular de cierre. En la versión no voraz de la expresión regular, Python busca la cadena más corta posible: `'<To serve man>'`. En la versión voraz, Python busca la cadena más larga posible: `'<To serve man> for dinner.>'`.

Buscar caracteres de salto de línea

El punto de `.` buscará cualquier cosa salvo un salto de línea. Pasando `re.DOTALL` como segundo argumento a `re.compile()`, podemos hacer que el punto busque todos los caracteres, incluido el salto de línea.

Escribe esto en el intérprete de comandos interactivo:

```
> > > import re
```

```
> > > no_newline_re = re.compile('.*')
```

```
> > > no_newline_re.search('Serve the public trust.\nProtect  
the innocent.
```

```
\nUphold the law.').group()
```

```
'Serve the public trust.'
```

```
> > > newline_re = re.compile('.*', re.DOTALL)
```

```
> > > newline_re.search('Serve the public trust.\nProtect the  
innocent.
```

```
\nUphold the law.').group()
```

```
'Serve the public trust.\nProtect the innocent.\nUphold the law.'
```

La expresión regular `no_newline_re`, que no ha pasado `re.DOTALL` a la llamada a `re.compile()` que la ha creado, buscará todo hasta el

primer salto de línea, mientras que `newline_re`, que sí ha pasado `re.DOTALL` a `re.compile()`, busca todo. Por eso, la llamada a `newline_re.search()` encuentra la cadena completa, incluidos los saltos de línea.

Buscar al principio y al final de una cadena

Podemos usar el símbolo de intercalación (^) al principio de una expresión regular para indicar que la coincidencia debe producirse al principio del texto buscado. Del mismo modo, podemos poner el símbolo del dólar (\$) al final de la expresión regular para indicar que la cadena debe terminar con ese patrón. Y podemos usar los dos símbolos a la vez para indicar que toda la cadena debe coincidir con la expresión regular; es decir, no basta con una coincidencia parcial con un subconjunto de la cadena.

Por ejemplo, la cadena de expresión regular `r'^Hello'` coincide con cadenas que empiezan por 'Hello'. Introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > import re
> > > begins_with_hello = re.compile(r'^Hello')
> > > begins_with_hello.search('Hello, world!')
<re.Match object; span = (0, 5), match = 'Hello'>
> > > begins_with_hello.search('He said "Hello."') = None
```

True

La cadena de expresión regular `r'\d$'` busca todas las cadenas que terminen con un carácter numérico entre 0 y 9. Escribe esto en el intérprete de comandos interactivo:

```
> > > import re
> > > ends_with_number = re.compile(r'\d$')
> > > ends_with_number.search('Your number is 42')
```

```
<re.Match object; span = (16, 17), match = '2'>
```

```
> > > ends_with_number.search('Your number is forty two.')  
= None
```

True

La cadena de expresión regular `r'^\d+$'` busca cadenas que empiecen y terminen con uno o varios caracteres numéricos. Pon esto en el intérprete de comandos:

```
> > > import re
```

```
> > > whole_string_is_num = re.compile(r'^\d+$')
```

```
> > > whole_string_is_num.search('1234567890')
```

```
<re.Match object; span = (0, 10), match = '1234567890'>
```

```
> > > whole_string_is_num.search('12345xyz67890') = None
```

True

Las dos últimas llamadas a `search()` en el ejemplo anterior muestran cómo la cadena entera debe coincidir con la expresión regular si usamos `^` y `$`.

También podemos usar `\b` para crear un patrón de expresión regular que coincida solo en un límite de palabra: el principio de una palabra, el final de una palabra o el principio y el final de una palabra. En este caso, una “palabra” es una secuencia de letras separadas por caracteres que no son letras. Por ejemplo, `r'\bcat.*?\b'` busca una palabra que empiece por ‘cat’ y vaya seguida de otros caracteres hasta el siguiente límite de palabra:

```
> > > import re
```

```
> > > pattern = re.compile(r'\bcat.*?\b')
```

```
> > > pattern.findall('The cat found a catapult catalog in the  
catacombs.')
```

['cat', 'catapult', 'catalog', 'catacombs']

La sintaxis `\B` busca cualquier cosa que no sea un límite de palabra:

```
> > > import re
```

```
> > > pattern = re.compile(r'\Bcat\B')
```

```
> > > pattern.findall('certificate') # Coincide
```

```
['cat']
```

```
> > > pattern.findall('catastrophe') # No coincide
```

```
[]
```

Esto es útil para buscar coincidencias en medio de una palabra.

Repaso de los símbolos de las expresiones regulares

En este capítulo hemos visto mucha notación ya, así que vamos a hacer un repaso rápido de lo que has aprendido sobre la sintaxis de las expresiones regulares básicas:

- ? busca cero o una apariciones del calificador precedente.
- * busca cero o más apariciones del calificador precedente.
- + busca una o varias apariciones del calificador precedente.
- {*n*} busca exactamente *n* apariciones del calificador precedente.
- {*n*,} busca *n* o más apariciones del calificador precedente.
- {*,m*} busca entre 0 y *m* apariciones del calificador precedente.
- {*n,m*} busca como mínimo *n* y como máximo *m* apariciones del calificador precedente.
- {*n,m*}? o *? o +? realiza una búsqueda no voraz del calificador precedente.
- ^spam significa que la cadena debe empezar por spam.

- `spam$` significa que la cadena debe terminar por `spam`.
- `.` busca cualquier carácter, excepto el salto de línea.
- `\d`, `\w` y `\s` buscan un número, una palabra o un carácter de espacio, respectivamente.
- `\D`, `\W` y `\S` buscan cualquier cosa salvo un dígito, una palabra o un espacio, respectivamente.
- `[abc` busca cualquier carácter entre los corchetes (como `a`, `b` o `c`).
- `^[abc` busca cualquier carácter que no esté entre los corchetes.
- `(Hello)` agrupa `'Hello'` como un solo calificador.

Búsquedas sin tener en cuenta mayúsculas y minúsculas

Por lo general, las expresiones regulares buscan texto con el uso exacto de mayúsculas/minúsculas que especificamos. Por ejemplo, las siguientes expresiones buscan cadenas completamente diferentes:

```
> > > import re
> > > pattern1 = re.compile('RoboCop')
> > > pattern2 = re.compile('ROBOCOP')
> > > pattern3 = re.compile('robOcop')
> > > pattern4 = re.compile('RobocOp')
```

Pero a veces lo que queremos es buscar las letras, sin importar si están en mayúscula o en minúscula. Para que a nuestra expresión regular le dé igual si son mayúsculas o minúsculas, podemos pasar `re.IGNORECASE` o `re.I` como segundo argumento a `re.compile()`. Escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > import re
> > > pattern = re.compile(r'robocop', re.I)
```

```
> > > pattern.search('RoboCop is part man, part machine, all  
cop.').group()
```

```
'RoboCop'
```

```
> > > pattern.search('ROBOCOP protects the  
innocent.').group()
```

```
'ROBOCOP'
```

```
> > > pattern.search('Have you seen robocop?').group()
```

```
'robocop'
```

La expresión regular ahora busca cadenas con mayúsculas o minúsculas.

Sustituir cadenas

Las expresiones regulares no se limitan a buscar patrones de texto; también pueden sustituir texto *in situ* en esos patrones. El método `sub()` para objetos `Pattern` acepta dos argumentos. El primero es una cadena que debería reemplazar cualquier coincidencia. El segundo es la cadena de expresión regular. El método `sub()` devuelve una cadena con las sustituciones hechas. Por ejemplo, escribe lo siguiente en el intérprete de comandos interactivo para cambiar los nombres de los agentes secretos por CENSORED:

```
> > > import re
```

```
> > > agent_pattern = re.compile(r'Agent \w+')
```

```
> > > agent_pattern.sub('CENSORED', 'Agent Alice contacted  
Agent Bob.')
```

```
'CENSORED contacted CENSORED.'
```

A veces, puede que necesitemos usar el propio texto coincidente como parte de la sustitución. En el primer argumento de `sub()`, podemos incluir `\1`, `\2`, `\3`, etc., para decir “introduce el texto del grupo 1, 2, 3, etc. en la sustitución”. Esta sintaxis se denomina “referencia inversa”.

Por ejemplo, supongamos que queremos censurar los nombres de los agentes secretos usando solo sus iniciales. Para hacerlo, podríamos usar la expresión regular `Agent (\w)\w*` y pasar `r'\1****'` como primer argumento a `sub()`:

```
> > > import re
> > > agent_pattern = re.compile(r'Agent (\w)\w*')
> > > agent_pattern.sub(r'\1****', 'Agent Alice contacted
Agent Bob.')
'A**** contacted B****.'
```

La cadena de expresión regular `\1` se sustituye por cualquier texto con el que coincida el grupo 1, es decir, el grupo `(\w)` de la expresión regular.

Gestionar expresiones regulares complejas con el modo verboso

Las expresiones regulares están bien si el patrón de texto que buscamos es sencillo, Pero buscar patrones complejos puede requerir expresiones regulares largas y enrevesadas. Podemos mitigar esa complejidad diciendo a la función `re.compile()` que ignore los espacios en blanco y comentarios dentro de la cadena de la expresión regular. Este “modo verboso” se habilita pasando la variable `re.VERBOSE` como segundo argumento a `re.compile()`. Ahora, en lugar de una expresión regular difícil de leer como esta

```
pattern = re.compile(r'((\d{3})|(\d{3}\.))?(\\s|-|\\.)?\\d{3}(\\s|-|\\.)\\d{4}(\\s*(ext|x|ext\\.\\s*\\d{2,5}))?')
```

podemos dividir la expresión regular en varias líneas y usar comentarios para etiquetar sus componentes, así:

```
pattern = re.compile(r'''(
\d{3}|(\d{3}\.))? # Prefijo de área
(\\s|-|\\.)? # Separador
```

`\d{3}` # Primeros tres dígitos

`(\s|-\|\. )` # Separador

`\d{4}` # Últimos cuatro dígitos

`(\s*(ext|x|ext\.)\s*\d{2,5})?` # Extensión

)””, re.VERBOSE)

Observa cómo el ejemplo anterior utiliza la sintaxis de comillas triples (""" para crear una cadena multilínea para poder repartir la definición de la expresión regular en varias líneas y hacerla mucho más legible.

Las reglas para los comentarios dentro de la cadena de expresión regular son iguales que para el código Python normal: el símbolo # y todo lo que hay detrás hasta el final de la línea se ignora. Además, los espacios adicionales dentro de la cadena multilínea de expresión regular no se consideran parte del patrón de texto que hay que buscar. Esto permite organizar las expresiones regulares de manera que sean más fáciles de leer. Aunque el modo verboso hace las cadenas de expresiones regulares más legibles, te recomiendo que uses en su lugar el módulo `Humre`, que veremos más adelante en este capítulo, para mejorar la legibilidad de tus expresiones regulares.

Combinar `re.IGNORECASE`, `re.DOTALL` y `re.VERBOSE`

¿Y si queremos usar `re.VERBOSE` para escribir comentarios en una expresión regular, pero también necesitamos usar `re.IGNORECASE` para no distinguir mayúsculas y minúsculas? Desafortunadamente, la función `re.compile()` solo toma un valor como segundo argumento. Podemos esquivar esta limitación combinando las variables `re.IGNORECASE`, `re.DOTALL` y `re.VERBOSE` con la barra vertical (`|`), que en este contexto se denomina “operador or bit a bit”. Por ejemplo, si queremos que una expresión regular no distinga mayúsculas y minúsculas e incluya saltos de línea para buscar el carácter punto, necesitaríamos una llamada a `re.compile()` como esta:

```
> > > some_regex = re.compile('foo', re.IGNORECASE |  
re.DOTALL)
```

Incluyendo las tres opciones en el segundo argumento queda así:

```
> > > some_regex = re.compile('foo', re.IGNORECASE |  
re.DOTALL | re.VERBOSE)
```

Esta sintaxis está un poco pasada de moda porque se originó en las primeras versiones de Python. Los detalles de los operadores bit a bit quedan fuera del ámbito de este libro. También podemos pasar otras opciones para el segundo argumento; no es lo habitual, pero puedes ver más sobre esto en los mismos recursos.

Proyecto 3: Extraer información de contacto de documentos grandes

Imagina que se te ha asignado la aburrida tarea de localizar todos los números de teléfono y direcciones de correo electrónico en un documento o una página web largos. Si te desplazas manualmente por la página, acabarás buscando mucho rato. Pero, si tuvieses un programa que pudiese buscar en el texto del portapapeles teléfonos y direcciones de correo, bastaría con pulsar **Control-A** para seleccionar todo el texto, **Control-C** para pegarlo en el portapapeles y ejecutar el programa. Podría reemplazar el texto del portapapeles por solo los teléfonos y direcciones que encuentre.

Siempre que se aborda un nuevo proyecto, puede resultar tentador lanzarse directamente a escribir código, pero, con mucha frecuencia, es mejor dar un paso atrás para tener una panorámica del asunto. Te recomiendo que primero hagas un plan de alto nivel para lo que necesita hacer tu programa. No pienses todavía en el código, ya te preocuparás por eso después. Ahora mismo, cñete a unas pinceladas.

Por ejemplo, el extractor de números de teléfono y direcciones de correo electrónico debe hacer lo siguiente:

- Coger el texto del portapapeles.
- Buscar todos los números de teléfono y direcciones de correo electrónico en el texto.

- Pegarlas en el portapapeles.

Ahora puedes empezar a pensar en cómo funcionaría esto con código. El código debe hacer lo siguiente:

- Usar el módulo `pyperclip` para copiar y pegar cadenas.
- Crear dos expresiones regulares, una para buscar teléfonos y otra para los correos.
- Buscar todas las coincidencias (no solo la primera) con ambas expresiones regulares.
- Organizar limpiamente las cadenas coincidentes en una única cadena que pegar.
- Mostrar un mensaje si no se han encontrado coincidencias en el texto.

Esta lista es la hoja de ruta del proyecto. Cuando escribas el código, puedes centrarte en cada uno de estos pasos por separado y cada paso debería ser bastante asequible, pues están expresados en términos de cosas que ya sabes hacer con Python.

Paso 1: crear una expresión regular para números de teléfono

Primero, hay que crear una expresión regular para buscar números de teléfono. Crea un archivo nuevo, escribe lo siguiente y guárdalo como `phoneAndEmail.py`:

```
import pyperclip, re

phone_re = re.compile(r'''(
    \d{3}|\(\d{3}\))? # Prefijo
    (\s|-|\.)? # Separador
    \d{3} # Primeros tres dígitos
    (\s|-|\.) # Separador
```

`(\d{4})` # Cuatro últimos dígitos

`(\s*(ext|x|ext\.)\s*(\d{2,5}))?` # Extensión

`)”`, re.VERBOSE)

PENDIENTE: Crear la expresión regular para el correo electrónico.

PENDIENTE: Buscar coincidencias en el texto del portapapeles.

PENDIENTE: Copiar los resultados en el portapapeles.

Los comentarios PENDIENTE son solo el esqueleto del programa, los iremos sustituyendo por el código real.

El número de teléfono empieza por un prefijo de área opcional, así que debemos poner un interrogante después del grupo. Dado que el prefijo solo puede tener tres dígitos (esto es, `\d{3}`) o tres dígitos entre paréntesis (es decir, `(\d{3})`), habría que usar una barra vertical para unir esas dos partes. Podemos añadir el comentario # Prefijo a esta parte para que la cadena multilínea nos ayude a recordar qué se supone que busca `(\d{3})|(\d{3})?`.

El carácter separador del número de teléfono puede ser un espacio (`\s`), un guion (`-`) o un punto (`.`) opcionales, así que deberíamos unir esas partes con barras verticales. Las siguientes partes de la expresión regular son fáciles: tres dígitos seguidos de otro separador, seguido de cuatro dígitos. La última parte es una extensión opcional compuesta de cualquier número de espacios seguidos por `ext`, `x` o `ext.`, seguido por entre dos y cinco dígitos.

Nota *Es fácil liarse al escribir expresiones regulares que contienen grupos con paréntesis () y paréntesis escapados \(\). Recuerda comprobar siempre que estás usando la sintaxis correcta si recibes un mensaje de error missing) o unterminated subpattern.*

Paso 2: crear una expresión regular para las direcciones de correo electrónico

También necesitarás una expresión regular que pueda buscar direcciones de correo electrónico. Da esta forma a tu programa:

```
import pyperclip, re
```

```
phone_re = re.compile(r'''(
```

```
--fragmento omitido--
```

```
# Crea la expresión regular para correo electrónico.
```

```
email_re = re.compile(r'''(
```

```
1 [a-zA-Z0-9._%+-] + # Nombre de usuario
```

```
2 @ # @ symbol
```

```
3 [a-zA-Z0-9.-] + # Nombre de dominio
```

```
(\.[a-zA-Z]{2,4}) # punto-algo
```

```
, re.VERBOSE)
```

```
# PENDIENTE: Buscar coincidencias en el texto del portapapeles.
```

```
# PENDIENTE: Copiar los resultados en el portapapeles.
```

La parte del nombre de usuario de la dirección de correo 1 consta de uno o varios caracteres, que pueden ser cualquiera de los siguientes: letras minúsculas y mayúsculas, números, un punto, un guion bajo, un signo de porcentaje, un signo más o un guion. Pueden ir todos en una clase de caracteres: `[a-zA-Z0-9._%+-]`.

El dominio y el nombre de usuario están separados por un símbolo `@` 2. El nombre del dominio 3 tiene una clase de caracteres un poco menos permisiva, con solo letras, números, puntos y guiones: `[a-zA-Z0-9.-]`. Por último, está la parte “punto com” (conocida técnicamente como dominio de nivel superior), que puede ser en realidad un punto y otra cosa. El formato para las direcciones de correo electrónico tiene muchas reglas raras. Esta expresión no encontrará cualquier dirección válida posible, pero sí prácticamente todas las direcciones típicas que haya.

Paso 3: buscar todas las coincidencias en el texto del portapapeles

Ahora que has especificado las expresiones regulares para teléfonos y direcciones de correo electrónico, puedes dejar que el módulo `re` de Python se ocupe de la ardua tarea de buscar todas las coincidencias que haya en el portapapeles. La función `pyperclip.paste()` te dará un valor de cadena del texto del portapapeles y el método `findall()` devolverá una lista de tuplas. Amplía así tu programa:

```
import pyperclip, re

phone_re = re.compile(r'''(
    --fragmento omitido--
# Busca coincidencias en el texto del portapapeles.

text = str(pyperclip.paste())

matches = []

for groups in phone_re.findall(text):

phone_num = '-'.join([groups[1], groups[3], groups[5]])

if groups[6] == :

phone_num = 'x' + groups[6]

matches.append(phone_num)

for groups in email_re.findall(text):

matches.append(groups)

# PENDIENTE: Copiar los resultados en el portapapeles.
```

Hay una tupla para cada coincidencia y cada tupla contiene cadenas para cada grupo de la expresión regular. Recuerda que el grupo 0 busca toda la expresión regular, así que el grupo en el índice 0 de la tupla es el que nos interesa.

Como ves en 1, las coincidencias se guardarán en una variable de lista llamada `matches`. Empieza como lista vacía y un par de bucles

for. Para las direcciones de correo, añadimos el grupo 0 de cada coincidencia 3. Para los números de teléfono, no tienes que añadir el grupo 0. Aunque el programa detecta números con varios formatos, queremos que se añada en un único formato estándar. La variable `phone_num` contiene una cadena formada a partir de los grupos 1, 3, 5 y 6 del texto coincidente 2. (Estos grupos son el prefijo, los primeros tres números, los cuatro últimos y la extensión).

Paso 4: unir las coincidencias en una cadena

Ahora que tienes los teléfonos y direcciones como lista de cadenas en `matches`, tienes que ponerlos en el portapapeles. La función `pyperclip.copy()` toma solo un valor de cadena, no una lista de cadenas, así que hay que llamar al método `join()` con `matches`.

Añade esto al programa:

```
import pyperclip, re

phone_re = re.compile(r'''(
    --fragmento omitido--

for groups in email_re.findall(text):
    matches.append(groups[0])

# Copia los resultados en el portapapeles.

if len(matches) > 0:
    pyperclip.copy('\n'.join(matches))
    print('Copied to clipboard:')
    print('\n'.join(matches))
else:
    print('No phone numbers or email addresses found.')
```

Para que sea más fácil ver que el programa funciona, vamos a imprimir también las coincidencias que encuentre en la ventana del terminal. Si no ha encontrado teléfonos ni direcciones, el programa se lo dice al usuario.

Para probar el programa, abre el navegador web y ve a la página de contactos de No Starch Press, en <https://nostarch.com/contactus>, pulsa **Control-A** para seleccionar todo el texto de la página y, después, **Control-C** para pegarlo en el portapapeles. Al ejecutar el programa, la salida debería ser algo así:

Copied to clipboard:

800-555-7240

415-555-9900

415-555-9950

info@nostarch.com

media@nostarch.com

academic@nostarch.com

info@nostarch.com

Puedes modificar este *script* para buscar direcciones de correo, redes sociales y muchos otros tipos de patrones.

Ideas para programas similares

Identificar patrones de texto (y posiblemente sustituirlos con el método `sub()`) tiene varias aplicaciones potenciales. Por ejemplo, podrías hacer lo siguiente:

- Buscar URL de sitios web que empiecen por `http://` o `https://`.
- Limpiar fechas en distintos formatos (como 3/14/2030, 03-14-2030 y 2030/3/14) sustituyéndolas por fechas en un único formato.

- Eliminar información sensible, como números de la Seguridad Social o de tarjetas de crédito.
- Buscar erratas habituales, como varios espacios entre palabras, palabras repetidas accidentalmente o varios signos de exclamación juntos.

Humre: un módulo para expresiones regulares legibles para humanos

El código se lee con mucha mayor frecuencia que con la que se escribe. Pero la densidad de puntuación de la sintaxis de las expresiones regulares puede ser difícil de leer incluso para programadores experimentados. Para resolver esto, el módulo de terceros Humre lleva las buenas ideas del modo verboso un paso más allá usando nombres legibles para humanos en inglés para crear código de expresiones regulares legible. Puedes instalar el módulo Humre siguiendo las instrucciones del apéndice A.

Volvamos al ejemplo del número de teléfono `r'\d{3}-\d{3}-\d{4}'` del principio del capítulo. Las funciones y constantes de Humre pueden producir la misma cadena de expresión regular con palabras en inglés normales:

```
> > > from humre import *
```

```
> > > phone_regex = exactly(3, DIGIT) + '-' + exactly(3,
DIGIT) + '-' + exactly(4, DIGIT)
```

```
> > > phone_regex
```

```
'\\d{3}-\\d{3}-\\d{4}'
```

Las constantes de Humre (como `DIGIT`) contienen cadenas y sus funciones (como `exactly()`) devuelven cadenas. Humre no sustituye al módulo `re`. Más bien, produce cadenas de expresión regular que podemos pasar a `re.compile()`:

```
> > > import re
```

```
> > > pattern = re.compile(phone_regex)
```

> > > **pattern.search('My number is 415-555-4242')**

<re.Match object; span = (13, 25), match = '415-555-4242'>

Humre tiene constantes y funciones para cada característica de la sintaxis de las expresiones regulares. Podemos concatenar las constantes y las cadenas devueltas como cualquier otra cadena. Por ejemplo, estas son las constantes de Humre para las clases de caracteres estenográficas:

- DIGIT** y **NONDIGIT** representan **r'\d'** y **r'\D'**, respectivamente.
- WORD** y **NONWORD** representan **r'\w'** y **r'\W'**, respectivamente.
- WHITESPACE** y **NONWHITESPACE** representan **r'\s'** y **r'\S'**, respectivamente.

Una fuente de errores habitual en las expresiones regulares es olvidar qué hay que usar con caracteres de escape. Podemos usar constantes de Humre en lugar de escribir los caracteres de escape nosotros. Por ejemplo, imagina que queremos buscar un número flotante de un solo dígito con un dígito también antes del punto decimal, como '0.9' o '4.5'. Si usamos la cadena de expresión regular **r'\d.\d'**, a lo mejor no nos damos cuenta de que el punto puede representar el decimal (como en '4.5'), pero también buscar cualquier otro carácter (como en '4A5').

En su lugar, usaremos la constante **PERIOD** de Humre, que contiene la cadena **r'\.'**. La expresión **DIGIT + PERIOD + DIGIT** se evalúa como **r'\d.\d'**, pero resulta mucho más obvio lo que está buscando la expresión regular.

Estas son las constantes de Humre para caracteres de escape:

~~PERIOD~~**PERIOD**
~~NONDIGIT~~**NONDIGIT**
~~NONWORD~~**NONWORD**
~~WHITESPACE~~**WHITESPACE**
~~NONWHITESPACE~~**NONWHITESPACE**

También hay constantes para el salto de línea, el tabulador y las comillas simples y dobles: **NEWLINE**, **TAB**, **QUOTE** y

DOUBLE_QUOTE. Las referencias inversas de `r'\1'` a `r'\99'` se representan como `BACK_1` a `BACK_99`.

No obstante, la mayor ganancia en legibilidad la obtendrás con las funciones de Humre. La tabla 9.2 muestra estas funciones y su equivalente en la sintaxis de expresiones regulares.

Tabla 9.2. Funciones de Humre.

Equivalencia de expresión regular

<code>group('A')</code>	<code>(A)</code>
<code>optional('A')</code>	<code>(A)?</code>
<code>either('A', 'B', 'C')</code>	<code>(A B C)</code>
<code>exactly(3, 'A')</code>	<code>A{3}</code>
<code>between(3, 5, 'A')</code>	<code>A{3,5}</code>
<code>at_least(3, 'A')</code>	<code>A{3,}</code>
<code>at_most(3, 'A')</code>	<code>A{,3}</code>
<code>chars('A-Z')</code>	<code>[A-Z]</code>
<code>non_chars('A-Z')</code>	<code>[^A-Z]</code>
<code>zero_or_more('A')</code>	<code>A*</code>
<code>zero_or_more_lazy('A')</code>	<code>A*?</code>
<code>one_or_more('A')</code>	<code>A+</code>
<code>one_or_more_lazy('A')</code>	<code>A+?</code>
<code>starts_with('A')</code>	<code>^A</code>
<code>ends_with('A')</code>	<code>A\$</code>
<code>starts_and_ends_with('A')</code>	<code>^A\$</code>
<code>named_group('name', 'A')</code>	<code>(?P<name>A)</code>

Humre también cuenta con varias funciones de conveniencia que combinan pares habituales de llamadas a funciones. Por ejemplo, en vez de usar `optional(group('A'))` para crear `'(A)?'`, podemos simplemente llamar a `optional_group('A')`. La tabla 9.3 recoge la lista completa de funciones de conveniencia de Humre.

Tabla 9.3. Funciones de conveniencia de Humre.

Equivalencia de conveniencia

<code>optional(group('A'))</code>	<code>optional_group('A')</code>
<code>group(either('A', 'B', 'C'))</code>	<code>group_either('A', 'B', 'C')</code>
<code>exactly(3, group('A'))</code>	<code>exactly_group(3, 'A')</code>
<code>between(3, 5, group('A'))</code>	<code>between_group(3, 5, 'A')</code>

```

re.findall(group(r'\A'))
re.findall(group(r'\A'))
zero_or_more(group('A'))
zero_or_more_lazy(group('A'))
one_or_more(group('A'))
one_or_more_lazy(group('A'))

```

Todas las funciones de Humre, excepto `either()` y `group_either()`, permiten pasar múltiples cadenas para unir las automáticamente. Esto significa que llamar a `group(DIGIT, PERIOD, DIGIT)` produce la misma cadena de expresión regular que `group(DIGIT + PERIOD + DIGIT)`. Ambas devuelven la cadena `r'(\d\.\d)'`.

Por último, Humre cuenta con constantes para patrones de expresiones regulares habituales:

- **ANY_SINGLE**: El patrón `.` que busca cualquier carácter (salvo saltos de línea).
- **ANYTHING_LAZY**: El patrón `.*?` cero o más perezoso.
- **ANYTHING_GREEDY**: El patrón `.*` cero o más voraz.
- **SOMETHING_LAZY**: El patrón `.+?` uno o más perezoso.
- **SOMETHING_GREEDY**: El patrón `.+` uno o más voraz.

La legibilidad de las expresiones regulares escritas con Humre se vuelve más evidente cuando te enfrentas a expresiones regulares complicadas y largas. Vamos a reescribir la expresión regular del número de teléfono del extractor de números de teléfono del proyecto anterior con Humre:

```

import re

from humre import *

phone_regex = group(
    optional_group(either(exactly(3, DIGIT), # Prefijo de área
    OPEN_PAREN + exactly(3, DIGIT) + CLOSE_PAREN)),

```

```

optional(group_either(WHITESPACE, '-', PERIOD)), # Separador
group(exactly(3, DIGIT)), # Tres primeros dígitos
group_either(WHITESPACE, '-', PERIOD), # Separador
group(exactly(4, DIGIT)), # Cuatro últimos dígitos
optional_group( # Extensión
zero_or_more(WHITESPACE),
group_either('ext', 'x', r'ext\.'),
zero_or_more(WHITESPACE),
group(between(2, 5, DIGIT))
)
)

pattern = re.compile(phone_regex)

match = pattern.search('My number is 415-555-1212.')

print(match.group())

```

Esta es la salida de este programa:

```
415-555-1212
```

Este código es mucho más verboso que el modo verboso de las expresiones regulares. Conviene importar Humre con la sintaxis `from humre import *` para no tener que poner `humre.` antes de cada función y constante, pero la longitud del código no es tan importante como la legibilidad.

Puedes convertir tus expresiones regulares en código de Humre llamando a la función `humre.parse()`, que devuelve una cadena de código fuente de Python:

```
> > > import humre
```

> > **humre.parse(r'\d{3}-\d{3}-\d{4}')**

“exactly(3, DIGIT) + '-' + exactly(3, DIGIT) + '-' + exactly(4, DIGIT)”

Cuando se combina con un editor moderno como PyCharm o Visual Studio Code, Humre ofrece varias ventajas adicionales:

- Se puede sangrar el código para dejar claro qué partes de la expresión regular contienen qué otras partes.
- El editor empareja correctamente los paréntesis.
- El editor resalta correctamente la sintaxis.
- El *linter* y las sugerencias de tipos de tu editor detectan errores tipográficos.
- La función de autocompletar del editor rellena los nombres de funciones y constantes.
- Humre gestiona las cadenas sin formato y el escape por ti.
- Puedes incluir comentarios de Python junto al código Humre.
- Los errores tipográficos generan mensajes de error más útiles.

Muchos programadores experimentados se niegan a usar cualquier cosa que no sea la sintaxis de expresiones regulares estándar, compleja e ilegible. Como programador, Peter Bhat Harkins dijo una vez: “Una de las cosas más irritantes que hacen habitualmente los programadores es sentirse tan bien por aprender algo difícil que no buscan formas de hacerlo más fácil, o incluso se oponen a cosas que lo harían”.

No obstante, si un compañero de trabajo se opone al uso de Humre, puedes sencillamente imprimir la cadena de expresión regular subyacente que genera tu código de Humre y ponerla otra vez en tu código fuente. Por ejemplo, el contenido de la variable `phone_regex` del proyecto del extractor de números de teléfono es así:

```
r'((\d{3})|(\d{3}))?(\s|-|\.)?(\d{3})(\s|-|\.)?(\d{4})(\s*(ext|x|ext
```

`\.)\s*(\d{2,5}))?)'`

Tu compañero puede usar esta cadena de expresión regular si lo considera más apropiado.

Resumen

Aunque un ordenador puede buscar texto deprisa, hay que decirle exactamente qué buscar. Las expresiones regulares permiten especificar el patrón de caracteres que queremos buscar en lugar del texto en sí. De hecho, algunas aplicaciones de procesamiento de texto y hojas de cálculo ofrecen funciones de búsqueda y reemplazo que permiten buscar con expresiones regulares. La sintaxis de las expresiones regulares, que depende mucho de la puntuación, se compone de calificadores que detallan qué buscar y cuantificadores que especifican cuánto buscar.

El módulo `re` que viene con Python permite compilar una cadena de expresión regular en un objeto `Pattern`. Estos objetos tienen varios métodos: `search()`, para buscar una sola coincidencia; `findall()`, para buscar todas las coincidencias; y `sub()`, para buscar y sustituir texto.

Encontrarás más información en la documentación oficial de Python, en <https://docs.python.org/3/library/re.html>. Otro recurso útil es el tutorial <https://www.regular-expressions.info>. La página de Humre en el Python Package Index es <https://pypi.org/project/Humre/>.

Preguntas prácticas

1. ¿Cuál es la función que devuelve objetos `Regex`?
2. ¿Por qué se utilizan a menudo cadenas crudas cuando se crean objetos `Regex`?
3. ¿Qué devuelve el método `search()`?
4. ¿Cómo obtenemos las cadenas reales que coinciden con el patrón de un objeto `Match`?
5. En la expresión regular creada con `r'(\d\d\d)-(\d\d\d-\d\d\d\d\d)'`,

¿qué cubre el grupo 0? ¿Y el grupo 1? ¿Y el 2?

6.Los paréntesis y los puntos tienen significados especiales en la sintaxis de las expresiones regulares. ¿Cómo especificamos que queremos que una expresión regular busque realmente paréntesis y puntos?

7.El método `findall()` devuelve una lista de cadenas o una lista de tuplas de cadenas. ¿Qué hace que devuelva una cosa o la otra?

8.¿Qué significa el carácter `|` en las expresiones regulares?

9.¿Qué dos cosas significa el carácter `?` en las expresiones regulares?

10.¿Cuál es la diferencia entre los caracteres `+` y `*` en las expresiones regulares?

11.¿Cuál es la diferencia entre `{3}` y `{3,5}` en las expresiones regulares?

12.¿Qué significan las clases de caracteres estenográficas `\d`, `\w` y `\s` en las expresiones regulares?

13.¿Qué significan las clases de caracteres estenográficas `\D`, `\W` y `\S` en las expresiones regulares?

14.¿Cuál es la diferencia entre las expresiones regulares `.*` y `.*??`?

15.¿Cuál es la sintaxis de clase de caracteres para buscar todos los números y letras en minúscula?

16.¿Cómo podemos hacer que una expresión regular no distinga entre mayúsculas y minúsculas?

17.¿Qué busca normalmente el carácter `?`? ¿Qué busca si pasamos `re.DOTALL` como segundo argumento a `re.compile()`?

18.Si `num_re = re.compile(r'\d+')`, ¿qué devolverá `num_re.sub('X', '12 drummers, 11 pipers, five rings, 3 hens')`?

19.¿Qué nos permite hacer pasar `re.VERBOSE` como segundo argumento a `re.compile()`?

Programas de práctica

Para practicar, escribe programas que realicen las siguientes tareas.

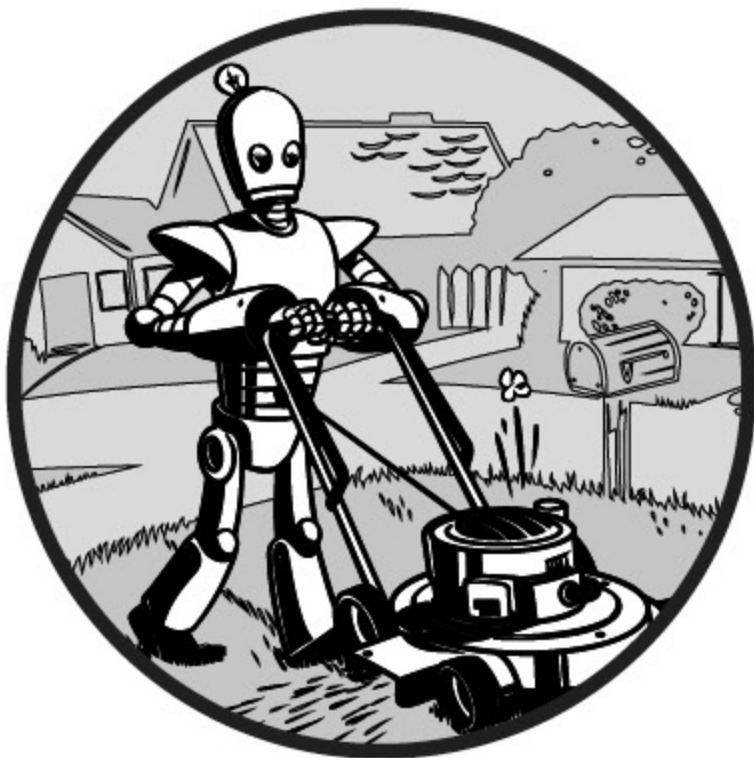
Detección de contraseñas fuertes

Escribe una función que utilice expresiones regulares para asegurarse de que la cadena de contraseña que se le pase sea fuerte. Una contraseña fuerte debe cumplir varias reglas: tener al menos ocho caracteres, contener mayúsculas y minúsculas y al menos un dígito. Pista: es más fácil probar la cadena con varios patrones de expresiones regulares que intentar dar con una única expresión regular que valide todas las reglas.

Versión en expresión regular del método strip()

Escribe una función que tome una cadena y haga lo mismo que el método de cadena strip(). Si no se pasan otros argumentos más que la cadena para quitar los espacios, la función debería eliminar los caracteres de espacio en blanco al principio y al final de la cadena. De lo contrario, debería eliminar los caracteres especificados en el segundo argumento.

Leer y escribir archivos



Las variables son una buena forma de almacenar datos mientras se ejecuta el programa, pero, si queremos que esos datos persistan una vez terminado el programa, necesitaremos guardarlos en un archivo. Imagina el contenido de un archivo como un único valor de cadena, que puede ocupar potencialmente varios gigabytes. En este capítulo, aprenderás a usar Python para crear, leer y guardar archivos en el disco duro.

Archivos y rutas de archivos

Un archivo tiene dos propiedades fundamentales: un nombre

(escrito por lo general como una sola palabra) y una ruta. La ruta especifica la ubicación de un archivo en el ordenador. Por ejemplo, en mi portátil Windows hay un archivo con el nombre `project.docx` en la ruta `C:\Users\Al\Documents`. La parte del nombre de archivo detrás del último punto se denomina “extensión” e indica qué tipo de archivo es. El archivo `project.docx` es un documento de Word y `Users`, `Al` y `Documents` son carpetas (o directorios). Las carpetas pueden contener archivos y otras carpetas (llamadas subcarpetas). Por ejemplo, `project.docx` está en la carpeta `Documents`, que está dentro de la carpeta `Al`, que a su vez está dentro de `Users`. La figura 10.1 muestra esta organización de carpetas.

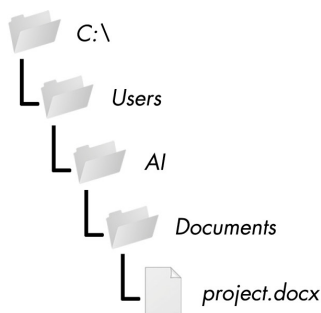


Figura 10.1. Un archivo en una jerarquía de carpetas.

La parte `C:\` de la ruta es el directorio raíz, que contiene todas las demás carpetas. En Windows, el directorio raíz se llama `C:\` y también se conoce como “unidad C:”. En macOS y Linux, el directorio raíz es `/`. En este libro, utilizaremos el estilo de Windows, `C:\`. Si estás introduciendo los ejemplos del intérprete de comandos interactivo en macOS o Linux, escribe `/` en su lugar.

Los volúmenes adicionales, como una unidad de DVD o de memoria USB, aparecerán de distinta manera en los diferentes sistemas operativos. En Windows, aparecen como nuevas unidades raíces con letras, como `D:\` o `E:\`. En macOS, aparecen como nuevas carpetas dentro de la carpeta `/Volúmenes`. En Linux, aparecen como nuevas carpetas dentro de la carpeta `/mnt`. Ten en cuenta que los nombres de los archivos y las carpetas no hacen distinción entre mayúsculas y minúsculas en Windows y macOS, pero sí en Linux.

Nota Como es probable que tu sistema tenga distintos archivos y

carpetas que el mío, no podrás seguir todos los ejemplos de este capítulo al pie de la letra, pero intenta hacerlo usando carpetas que existan en tu ordenador.

Estandarizar los separadores de ruta

En Windows, las rutas se escriben usando barras invertidas (\) como separador entre nombres de carpetas. En cambio, los sistemas operativos macOS y Linux utilizan la barra oblicua (/) como separador de ruta.

La función Path() del módulo pathlib maneja todos los sistemas operativos, así que lo mejor es usar barras oblicuas en nuestro código Python. Si le pasamos los valores de cadena de los nombres individuales del archivo y las carpetas de la ruta, Path() devolverá una cadena con una ruta que utiliza los separadores correctos. Escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > from pathlib import Path
```

```
> > > Path('spam', 'bacon', 'eggs')
```

```
WindowsPath('spam/bacon/eggs')
```

```
> > > str(Path('spam', 'bacon', 'eggs'))
```

```
'spam\\bacon\\eggs'
```

Aunque el objeto WindowsPath pueda usar barras oblicuas, convertirlo a una cadena con la función str() requiere usar barras invertidas. Observa que la convención para importar pathlib es ejecutar from pathlib import Path, ya que de otro modo habría que escribir pathlib.Path cada vez que aparezca Path en el código. Ese tecleo extra es redundante. Estoy ejecutando los ejemplos del intérprete de comandos interactivo de este capítulo en Windows, por eso Path('spam', 'bacon', 'eggs') devuelve un objeto WindowsPath para la ruta unida, representada como WindowsPath('spam/bacon/eggs'). Aunque Windows usa barras invertidas, la representación WindowsPath en el intérprete de comandos interactivo las muestra como barras oblicuas, ya que los desarrolladores de código abierto han favorecido históricamente el

sistema operativo Linux.

Si queremos obtener una cadena de texto simple de esta ruta, podemos pasarla a la función `str()`, que en nuestro ejemplo devuelve `'spam\\bacon\\eggs'`. (Observa que duplicamos las barras invertidas porque necesitamos convertir cada una en un carácter de escape). Si hubiese llamado a esta función en macOS o Linux, `Path()` habría devuelto un objeto `PosixPath` que, al pasarse a `str()`, habría devuelto `'spam/bacon/eggs'`. (POSIX es un conjunto de normas para los sistemas operativos de tipo Unix).

Si trabajas con objetos `Path`, `WindowsPath` y `PosixPath` no tienen que aparecer nunca directamente en tu código fuente. Estos objetos `Path` se pasarán a muchas de las funciones relacionadas con el archivo que veremos en este capítulo. Por ejemplo, el siguiente código une nombres de una lista de nombres de archivo al final del nombre de una carpeta:

```
> > > from pathlib import Path

> > > my_files = ['accounts.txt', 'details.csv', 'invite.docx']

> > > for filename in my_files:

... print(Path(r'C:\Users\Al', filename))

.

C:\Users\Al\accounts.txt

C:\Users\Al\details.csv

C:\Users\Al\invite.docx
```

En Windows, la barra invertida separa directorios, así que no podemos usarla en nombres de archivo. En cambio, sí podemos hacerlo en macOS y Linux. Así pues, mientras que `Path(r'spam\eggs')` hace referencia a dos carpetas separadas (o a un archivo `eggs` en una carpeta `spam`) en Windows, el mismo comando se referiría a un único archivo (o carpeta) llamado `spam\eggs` en macOS y Linux. Por este motivo, suele ser buena idea usar siempre barras oblicuas en el código Python (es lo que voy a hacer durante

el resto del capítulo). El módulo `pathlib` se asegurará de que tu código funcione en todos los sistemas operativos.

Unir rutas

Por lo general, usamos el operador `+` para sumar dos números enteros o flotantes, como en la expresión `2 + 2`, que se evalúa al valor 4. Pero también podemos usar este operador para concatenar dos valores de cadena, como la expresión `'Hello' + 'World'`, que se evalúa al valor de cadena `'HelloWorld'`. Del mismo modo, el operador `/` que normalmente usamos para la división puede combinar objetos `Path` y cadenas. Esto es útil para modificar un objeto `Path` después de haberlo creado con la función `Path()`.

Para ver un ejemplo, escribe esto en el intérprete de comandos interactivo:

```
> > > from pathlib import Path
```

```
> > > Path('spam') / 'bacon' / 'eggs'
```

```
WindowsPath('spam/bacon/eggs')
```

```
> > > Path('spam') / Path('bacon/eggs')
```

```
WindowsPath('spam/bacon/eggs')
```

```
> > > Path('spam') / Path('bacon', 'eggs')
```

```
WindowsPath('spam/bacon/eggs')
```

Lo único que debes tener en mente cuando uses el operador `/` para unir rutas es que uno de los primeros dos valores de la expresión debe ser un objeto `Path`, ya que estas expresiones se evalúan de izquierda a derecha y el operador se puede usar con dos objetos `Path` o uno y una cadena, pero no con dos cadenas. Python te dará error si intentas introducir lo siguiente en el intérprete de comandos interactivo:

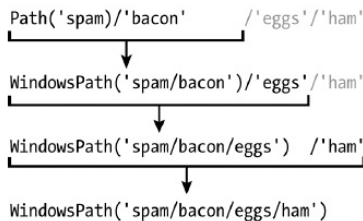
```
> > > 'spam' / 'bacon'
```

```
Traceback (most recent call last):
```

File “<python-input-0>”, line 1, in <module>

TypeError: unsupported operand type(s) for /: ‘str’ and ‘str’

Así pues, o el primero o el segundo valor de la izquierda debe ser un objeto Path para que toda la expresión se evalúe a un objeto Path. Así es como el operador / y un objeto Path se evalúan como el objeto Path final:



Si ves el mensaje de error `TypeError: unsupported operand type(s) for /: ‘str’ and ‘str’` de antes, tienes que poner un objeto Path en lugar de una cadena a la izquierda de la expresión.

El operador / sustituye a la antigua función `os.path.join()`, sobre la que puedes aprender más en <https://docs.python.org/3/library/os.path.html#os.path.join>.

Acceder al directorio de trabajo actual

Todo programa que se ejecuta en tu ordenador tiene un directorio de trabajo actual. Se asume que cualquier nombre de archivo o ruta que no empiece con el directorio raíz está en el directorio actual.

Nota Aunque “carpeta” es el nombre más moderno para “directorio”, observa que *directorio de trabajo actual* (o *directorio de trabajo*) es el término habitual.

Podemos obtener el directorio de trabajo actual con la función `Path.cwd()` y cambiarlo con `os.chdir()`. Escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > from pathlib import Path
```

```
> > > import os
```

```
> > > Path.cwd()
```

```
WindowsPath('C:/Users/Al/AppData/Local/Programs/Python/Python313')
```

```
> > > os.chdir('C:\\Windows\\System32')
```

```
> > > Path.cwd()
```

```
WindowsPath('C:/Windows/System32')
```

Aquí, el directorio de trabajo actual es C:\Users\Al\AppData\Local\Programs\Python\Python313, así que el nombre de archivo project.docx se refiere a C:\Users\Al\AppData\Local\Programs\Python\Python313\project.docx. Cuando cambiamos el directorio de trabajo actual a C:\Windows\System32, the filename project.docx se interpreta como C:\Windows\System32\project.docx.

Python dará error si intentamos cambiar a un directorio que no existe:

```
> > > import os
```

```
> > > os.chdir('C:/ThisFolderDoesNotExist')
```

Traceback (most recent call last):

File "<python-input-0>", line 1, in <module>

FileNotFoundError: [WinError 2] The system cannot find the file specified:

```
'C:/ThisFolderDoesNotExist'
```

No hay función pathlib para cambiar el directorio de trabajo. Hay que usar os.chdir().

La función os.getcwd() es la manera antigua de obtener el directorio de trabajo actual como una cadena. Está documentada en <https://>

docs.python.org/3/library/os.html#os.getcwd.

Acceder a la carpeta de inicio

Todos los usuarios tienen una carpeta para sus archivos en el ordenador; esta carpeta se llama directorio de inicio o carpeta de inicio. Podemos obtener un objeto Path de la carpeta de inicio llamando a Path.home():

```
> > > from pathlib import Path
```

```
> > > Path.home()
```

```
WindowsPath('C:/Users/Al')
```

Las carpetas de inicio se encuentran en un lugar predeterminado, dependiendo del sistema operativo:

- En Windows, los directorios de inicio están en C:\Usuarios.
- En macOS, en /Usuarios.
- En Linux, suelen estar en /home.

Es casi seguro que tus *scripts* tendrán permiso para leer y escribir en los archivos de tu carpeta de inicio, así que es un lugar ideal para poner los archivos de los programas de Python con los que trabajas.

Especificar rutas absolutas o relativas

Hay dos maneras de especificar una ruta de archivo:

- Una ruta absoluta, que empieza siempre con el directorio raíz (C:\ en Windows y / en macOS y Linux)
- Una ruta relativa, que es relativa al directorio de trabajo actual del programa.

En Windows, C:\ es la raíz de la unidad principal. Esta denominación con letras se remonta a los años sesenta, cuando los ordenadores tenían dos disqueteras llamadas A:\ y B:\. En

Windows, se asigna a las unidades de memoria flash USB y las de DVD letras a partir de D:\. Usa una de estas unidades como directorio raíz para acceder a archivos en esos medios de almacenamiento.

También están las carpetas . y ... No son carpetas de verdad, sino nombres especiales que se pueden usar en una ruta de archivo. Un solo punto para un nombre de carpeta es un atajo para esta carpeta. Dos puntos significan su carpeta primaria.

La figura 10.2 muestra archivos y carpetas de ejemplo. Cuando el directorio de trabajo actual es C:\bacon, las rutas relativas para los demás archivos y carpetas son como se muestra en la figura.

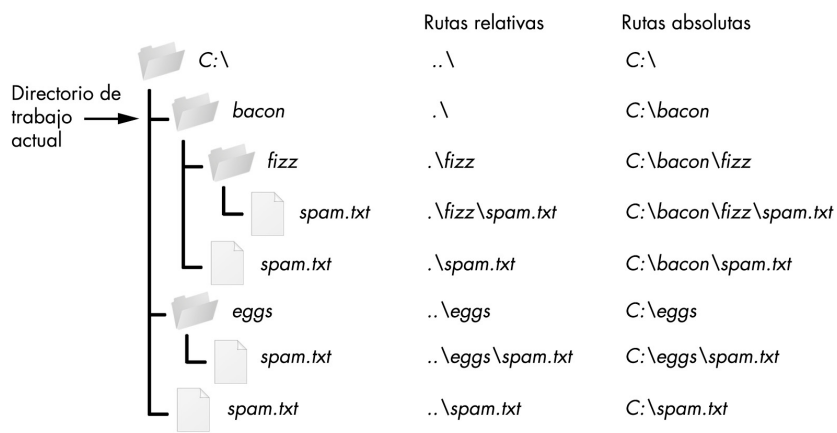


Figura 10.2. Las rutas relativas de archivos y carpetas en el directorio de trabajo actual C:\bacon.

El .\ al principio de una ruta relativa es opcional. Por ejemplo, .\spam.txt y spam.txt se refieren al mismo archivo.

Crear carpetas nuevas

Nuestros programas pueden crear carpetas nuevas con la función `os.makedirs()`. Introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > import os
```



```
> > > os.makedirs('C:\\delicious\\walnut\\waffles')
```

Esto creará la carpeta C:\delicious y una carpeta walnut dentro de ella y una carpeta waffles dentro de C:\delicious\walnut. Es decir, os.makedirs() creará cualquier carpeta intermedia necesaria para asegurarse de que la ruta completa existe. La figura 10.3 muestra esta jerarquía de carpetas.

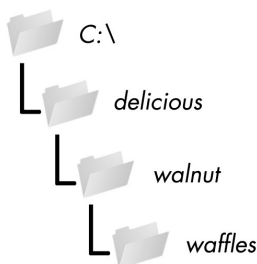


Figura 10.3. El resultado de os.makedirs('C:\\delicious\\walnut\\waffles').

Para crear una carpeta a partir de un objeto Path, llama al método mkdir(). Por ejemplo, este código creará una carpeta spam en la carpeta de inicio de mi ordenador:

```
> > > from pathlib import Path
```

```
> > > Path(r'C:\Users\Al\spam').mkdir()
```

Observa que mkdir() solo puede crear una carpeta, a menos que le pasemos parents=True, en cuyo caso crea también todas las carpetas primarias necesarias.

Manejar rutas absolutas y relativas

Llamar al método is_absolute() con un objeto Path devolverá True si representa una ruta absoluta o False si representa una ruta relativa. Por ejemplo, escribe esto en el intérprete de comandos interactivo, usando tus propios archivos y carpetas en lugar de los que pongo yo aquí:

```
> > > from pathlib import Path
```

```
> > > Path.cwd()
```

```
WindowsPath('C:/Users/Al/AppData/Local/Programs/Python/Python312')
```

```
> > > Path.cwd().is_absolute()
```

```
True
```

```
> > > Path('spam/bacon/eggs').is_absolute()
```

```
False
```

Para obtener una ruta absoluta a partir de una relativa, puedes poner `Path.cwd()` / delante del objeto `Path` relativo. Después de todo, cuando decimos “ruta relativa”, casi siempre queremos decir una ruta relativa al directorio de trabajo actual. El método `absolute()` también devuelve este objeto `Path`. Escribe esto en el intérprete de comandos interactivo:

```
> > > from pathlib import Path
```

```
> > > Path('my/relative/path')
```

```
WindowsPath('my/relative/path')
```

```
> > > Path.cwd() / Path('my/relative/path')
```

```
WindowsPath('C:/Users/Al/Desktop/my/relative/path')
```

```
> > > Path('my/relative/path').absolute()
```

```
WindowsPath('C:/Users/Al/Desktop/my/relative/path')
```

Si la ruta relativa es relativa a otra ruta que no sea el directorio de trabajo actual, sustituye `Path.cwd()` por esa otra ruta. El siguiente ejemplo obtiene una ruta absoluta utilizando la carpeta de inicio en vez del directorio de trabajo actual:

```
> > > from pathlib import Path
```

```
> > > Path('my/relative/path')
```

WindowsPath('my/relative/path')

> > **Path.home() / Path('my/relative/path')**

WindowsPath('C:/Users/Al/my/relative/path')

Los objetos Path sirven para representar tanto rutas relativas como absolutas. La única diferencia está en si el objeto Path empieza con el directorio raíz o no.

Obtener las partes de una ruta de archivo

Con un objeto Path dado, podemos extraer las distintas partes de la ruta utilizando varios atributos del objeto Path. Esto puede ser útil para construir nuevas rutas basadas en otras existentes. La figura 10.4 ofrece un diagrama de los atributos.

Las partes de una ruta de archivo incluyen lo siguiente:

- El ancla, que es el directorio raíz del sistema de archivos.
- En Windows, la unidad, que es la letra que denota una unidad física u otro dispositivo de almacenamiento.
- La carpeta primaria, que es la que contiene el archivo.
- El nombre del archivo, formado por el nombre base y un sufijo (o extensión).

Observa que los objetos Path de Windows tienen un atributo drive que no tienen estos objetos en macOS y Linux. El atributo drive no incluye la primera barra invertida.

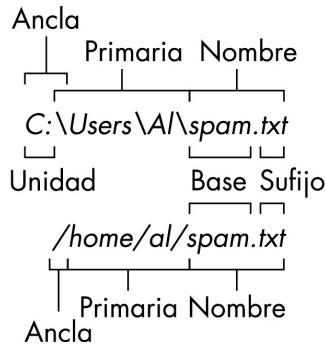


Figura 10.4. Las partes de una ruta de archivo en Windows (arriba) y macOS/Linux (abajo).

Para extraer cada atributo de la ruta de archivo, escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > from pathlib import Path
> > > p = Path('C:/Users/Al/spam.txt')
> > > p.anchor
'C:\\'
> > > p.parent
WindowsPath('C:/Users/Al')
> > > p.name
'spam.txt'
> > > p.stem
'spam'
> > > p.suffix
'.txt'
> > > p.drive
```

'C:.'

Estos atributos se evalúan como valores de cadena sencillos, salvo `parent`, que se evalúa como otro objeto `Path`. Si quieres descomponer una ruta por separado, accede al atributo `parts` para obtener una tupla de valores de cadena:

```
> > > from pathlib import Path
> > > p = Path('C:/Users/Al/spam.txt')
> > > p.parts
('C:\\', 'Users', 'Al', 'spam.txt')
> > > p.parts[3]
'spam.txt'
> > > p.parts[0:2]
('C:\\', 'Users')
```

Observa que, aunque la cadena usada en la llamada a `Path()` contiene barras oblicuas, `parts` utiliza un ancla en Windows que tiene las barras invertidas adecuadas: `'C:\\'` (o `r'C:\\'` como cadena cruda sin caracteres de escape).

El atributo `parents` (que no es lo mismo que `parent`) se evalúa a las carpetas primarias de un objeto `Path` con un índice entero:

```
> > > from pathlib import Path
> > > Path.cwd()
WindowsPath('C:/Users/Al/Desktop')
> > > Path.cwd().parents[0]
WindowsPath('C:/Users/Al')
> > > Path.cwd().parents[1]
```

```
WindowsPath('C:/Users')
```

```
> > > Path.cwd().parents [2]
```

```
WindowsPath('C:/')
```

Si continuásemos siguiendo las carpetas primarias, llegaríamos al directorio raíz.

Encontrar tamaños de archivo y sellos de tiempo

Una vez que ya conoces formas de manejar rutas de archivo, puedes empezar a reunir información sobre archivos y carpetas específicos. El método `stat()` devuelve un objeto `stat_result` con información sobre el tamaño del archivo y un sello de tiempo. A modo de ejemplo, introduce lo siguiente en el intérprete de comandos interactivo para saber más sobre el archivo de programa `calc.exe` e Windows:

```
> > > from pathlib import Path
```

```
> > > calc_file = Path('C:/Windows/System32/calc.exe')
```

```
> > > calc_file.stat()
```

```
os.stat_result(st_mode=33279, st_ino=562949956525418,  
st_dev=3739257218,
```

```
st_nlink=2, st_uid=0, st_gid=0, st_size=27648,  
st_atime=1678984560,
```

```
st_mtime=1575709787, st_ctime=1575709787)
```

```
> > > calc_file.stat().st_size
```

```
27648
```

```
> > > calc_file.stat().st_mtime
```

```
1712627129.0906117
```

```
> > > import time
```

> > **time.asctime(time.localtime(calc_file.stat().st_mtime))**

'Mon Apr 8 20:45:29 2024'

El atributo `st_size` del objeto `stat_result` devuelto por el método `stat()` es el tamaño de archivo en bytes. Podemos dividir este entero por 1024, `1024 ** 2` o `1024 ** 3` para obtener el tamaño en KB, MB o GB, respectivamente. `st_mtime` es el sello de tiempo de la última modificación, que puede ser útil para saber, por ejemplo, cuándo se modificó por última vez un archivo .docx de Word. Este sello de tiempo está en tiempo Epoch de Unix, que es el número de segundos transcurridos desde el 1 de enero de 1970. El módulo `time` (explicado en el capítulo 19) cuenta con funciones para convertir este número a un formato legible para humanos.

El objeto `stat_result` tiene varios atributos útiles:

- `st_size`: El tamaño del archivo en bytes.
- `st_mtime`: El sello de tiempo de la última modificación realizada en el archivo.
- `st_ctime`: El sello de tiempo de la creación del archivo. En Windows, esto identifica cuándo se creó el archivo; en macOS y Linux, la última vez que cambiaron los metadatos del archivo (por ejemplo, el nombre).
- `st_atime`: El sello de tiempo del último acceso, cuando se leyó el archivo por última vez.

Ten en cuenta que los sellos de tiempo de modificación, creación y acceso se pueden cambiar manualmente y no hay garantías de que sean exactos.

Buscar archivos con patrones globales

Podemos usar los caracteres `*` y `?` para buscar nombres de carpetas y archivos en los denominados patrones globales. Estos patrones son como un lenguaje de expresiones regulares simplificado: `*` busca cualquier texto y `?` solo un carácter exactamente. Fíjate en estos patrones globales:

- `*.txt`: Busca todos los archivos que terminan en `.txt`.
- `project?.txt`: Busca `project1.txt`, `project2.txt`, o `projectX.txt`.
- `*project?.*`: Busca `catproject5.txt` o `secret_project7.docx`.
- `*`: Busca todos los nombres de archivo.

Los objetos `Path` de carpetas tienen un método `glob()` para enumerar cualquier contenido en la carpeta que coincida con el patrón global. El método `glob()` devuelve un objeto generador (tema que queda fuera del alcance de este libro) que habrá que pasar a `list()` para ver fácilmente en el intérprete de comandos interactivo:

```
> > > from pathlib import Path
```

```
> > > p = Path('C:/Users/Al/Desktop')
```

```
> > > p.glob('*')
```

```
<generator object Path.glob at 0x000002A6E389DED0 >
```

```
> > > list(p.glob('*'))
```

```
[WindowsPath('C:/Users/Al/Desktop/1.png'), WindowsPath('C:/Users/Al/
```

```
Desktop/22-ap.pdf'), WindowsPath('C:/Users/Al/Desktop/cat.jpg'),
```

```
WindowsPath('C:/Users/Al/Desktop/zzz.txt')]
```

También se puede usar el objeto generador que devuelve `glob()` en un bucle `for`:

```
> > > from pathlib import Path
```

```
> > > for name in Path('C:/Users/Al/Desktop').glob('*'):
```

```
> > > print(name)
```

```
C:\Users\Al\Desktop\1.png
```

```
C:\Users\Al\Desktop\22-ap.pdf
```



```
C:\Users\Al\Desktop\cat.jpg
```

```
C:\Users\Al\Desktop\zzz.txt
```

Si queremos realizar una operación con todos los archivos de una carpeta, como hacer una copia de seguridad o renombrarlos, la llamada al método `glob('*')` puede darnos una lista de objetos `Path` para estos archivos y carpetas. Observa que los patrones globales también se usan habitualmente en comandos de línea de comandos como `ls` o `dir`. El capítulo 12 explica con más detalle la línea de comandos.

Comprobar la validez de una ruta

Muchas funciones de Python darán error si les damos una ruta que no existe. Por suerte, los objetos `Path` cuentan con métodos para comprobar si una ruta dada existe y si es un archivo o una carpeta. Suponiendo que una variable `p` contenga un objeto `Path`, podríamos esperar lo siguiente:

- Llamar a `p.exists()` devuelve `True` si la ruta existe y `False` si no existe.
- Llamar a `p.is_file()` devuelve `True` si la ruta existe y es un archivo y `False` en caso contrario.
- Llamar a `p.is_dir()` devuelve `True` si la ruta existe y es un directorio y `False` en caso contrario.

En mi ordenador, esto es lo que me sale cuando pruebo estos métodos en el intérprete de comandos interactivo:

```
> > > from pathlib import Path
> > > win_dir = Path('C:/Windows')
> > > not_exists_dir = Path('C:/This/Folder/Does/Not/Exist')
> > > calc_file_path = Path('C:/Windows
/System32/calc.exe')
```

```
> > > win_dir.exists()
```

True

```
> > > win_dir.is_dir()
```

True

```
> > > not_exists_dir.exists()
```

False

```
> > > calc_file_path.is_file()
```

True

```
> > > calc_file_path.is_dir()
```

False

Podemos determinar si hay una unidad de DVD o flash conectada ahora mismo al ordenador buscándola con el método `exists()`. Por ejemplo, si quisiera comprobar una unidad flash con el volumen D:\ de mi ordenador Windows, podría hacerlo así:

```
> > > from pathlib import Path
```

```
> > > d_drive = Path('D:/')
```

```
> > > d_drive.exists()
```

False

¡Vaya! Parece que se me ha olvidado conectar mi unidad flash.

El antiguo módulo `os.path` puede hacer la misma tarea con las funciones `os.path.exists(ruta)`, `os.path.isfile(ruta)` y `os.path.isdir(ruta)`, que actúan como sus equivalentes de la función `Path`. En Python 3.6, estas funciones pueden aceptar objetos `Path` además de cadenas de las rutas de archivo.

El proceso de lectura y escritura de archivos

Cuando estés cómodo trabajando con carpetas y rutas relativas, podrás especificar las ubicaciones de archivos para leer y escribir. Las funciones de las que hablaremos en los próximos apartados se aplican a archivos de texto simple. Estos archivos contienen solo caracteres de texto básicos y no incluyen información sobre la fuente, el tamaño o el color. Los archivos de texto con la extensión .txt o los *scripts* de Python con la extensión .py son ejemplos de archivos de texto simple. Podemos abrir estos archivos con el Bloc de notas de Windows o la aplicación TextEdit de macOS, y los programas pueden leer fácilmente su contenido, así que los trataremos como un valor de cadena normal.

Los archivos binarios son otro tipo de archivo, como los documentos de procesadores de texto, los PDF, las imágenes, las hojas de cálculo y los programas ejecutables. Si abrimos un archivo binario en el Bloc de notas o en TextEdit, parecerá una mezcla de caracteres sin sentido, como lo que vemos en la figura 10.5.

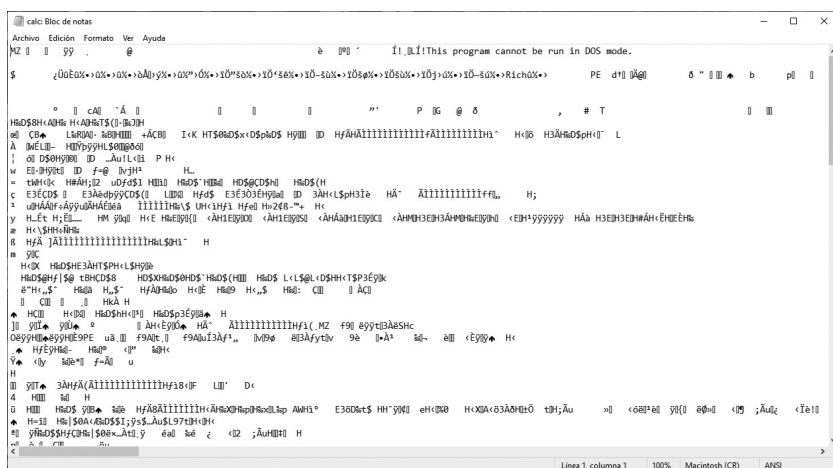


Figura 10.5. El programa calc.exe de Windows abierto en el Bloc de notas.

Dado que hay que manejar cada archivo de tipo binario a su manera, este libro no abordará cómo leer y escribir archivos binarios directamente. Por suerte, muchos módulos facilitan el trabajo con estos archivos y, más adelante en el capítulo, exploraremos uno de ellos: *shelve*.

El método `read_text()` del módulo `pathlib` devuelve el contenido completo de un archivo de texto como una cadena. Su método `write_text()` crea un nuevo archivo de texto (o sobrescribe uno que ya existe) con la cadena que se le pasa. Escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > from pathlib import Path
```

```
> > > p = Path("spam.txt")
```

```
> > > p.write_text('Hello, world!')
```

```
13
```

```
> > > p.read_text()
```

```
'Hello, world!'
```

Estas llamadas crean un archivo `spam.txt` con el contenido `'Hello, world!'`. El 13 que devuelve `write_text()` indica que se han escrito 13 caracteres en el archivo. (A menudo se puede ignorar este valor de retorno). La llamada a `read_text()` lee y devuelve el contenido del nuevo archivo como una cadena: `'Hello, world!'`.

Ten en cuenta que estos métodos del objeto `Path` permiten solo interacciones básicas con los archivos. La forma más habitual de escribir en un archivo implica usar la función `open()` y objetos de archivo. Hay tres pasos para leer o escribir archivos en Python:

- 1.Lllamar a la función `open()` para devolver un objeto `File`.
- 2.Lllamar al método `read()` o `write()` con el objeto `File`.
- 3.Cerrar el archivo llamando al método `close()` con el objeto `File`.

Veremos estos pasos en los siguientes apartados.

Ten en cuenta que, al empezar a trabajar con archivos, puede resultarte útil ser capaz de ver rápidamente sus extensiones (`.txt`, `.pdf`, `.jpg`, etc.). Windows y macOS ocultan por defecto las extensiones de archivo y muestran `spam.txt` como `spam`. Para ver las extensiones, abre los ajustes del Explorador de archivos (en

Windows) o Finder (en macOS) y busca una casilla de verificación que diga algo como Mostrar todas las extensiones de nombre de archivo (la ubicación y redacción exacta de esta opción dependerá de la versión del sistema operativo).

Abrir archivos

Para abrir un archivo con la función `open()`, pásale una ruta de cadena indicando el archivo que quieres abrir. Puede ser una ruta absoluta o relativa. La función `open()` devuelve un objeto `File`.

Prueba creando un archivo de texto llamado `hello.txt` con el Bloc de notas o TextEdit. Escribe `Hello, world!` Como contenido del archivo y guárdalo en tu carpeta de inicio. A continuación, introduce esto en el intérprete de comandos interactivo:

```
> > > from pathlib import Path  
  
> > > hello_file = open(Path.home() / 'hello.txt',  
encoding='UTF-8')
```

La función `open()` abrirá el archivo en modo lectura de texto simple o modo lectura para abreviar. Cuando un archivo se abre en modo lectura, Python permite leer los datos del archivo, pero no escribir ni modificarlo de ninguna manera. El modo lectura es el predeterminado para los archivos que abrimos en Python, pero, si no quieres depender de los predeterminados de Python, puedes especificar explícitamente el modo pasando el valor `'r'` como segundo argumento a `open()`. Por ejemplo, `open('/Users/Al/hello.txt', 'r')` hace lo mismo que `open('/Users/Al/hello.txt')`.

El parámetro `encoding` especifica qué codificación usar al convertir los bytes del archivo en una cadena de texto de Python. La codificación correcta es casi siempre `'utf-8'`, que también es la predeterminada en macOS y Linux. Sin embargo, Windows utiliza `'cp1252'` como codificación predeterminada (también conocida como ASCII extendido). Esto puede dar problemas cuando se intenta leer determinados archivos de texto codificados en UTF-8 con caracteres no latinos en Windows, así que es una buena costumbre pasar `encoding='utf-8'` a las llamadas a la función `open()` cuando se abren textos en modo lectura, escritura o agregar.

Los modos leer, escribir y agregar binarios no utilizan el parámetro encoding, así que se puede prescindir de él en esos casos.

La llamada a `open()` devuelve un objeto `File`. Estos objetos representan archivos en nuestro ordenador; es solo otro tipo de valor en Python, como las listas y los diccionarios con los que ya estás familiarizado. En el ejemplo anterior, hemos guardado el objeto `File` en la variable `hello_file`. Ahora, siempre que queramos leer de ese archivo o escribir en él, podemos hacerlo llamando a métodos con el objeto `File` en `hello_file`.

Leer el contenido de archivos

Ahora que tenemos un objeto `File`, podemos empezar a leer de él. Si queremos leer todo el contenido de un archivo como valor de cadena, utilizaremos el método `read()` del objeto `File`. Continuemos con el ejemplo del objeto `File` `hello.txt` almacenado en `hello_file`. Escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > hello_content = hello_file.read()
```

```
> > > hello_content
```

```
'Hello, world!'
```

Imagina el contenido de un archivo como un único valor de cadena grande; el método `read()` simplemente devuelve la cadena almacenada en el archivo.

Como alternativa, podemos usar el método `readlines()` para obtener una lista de valores de cadena del archivo, uno por cada línea de texto. Por ejemplo, crea un archivo llamado `sonnet29.txt` en la misma carpeta que `hello.txt` y escribe dentro el siguiente texto:

When, in disgrace with fortune and men's eyes,

I all alone beweep my outcast state,

And trouble deaf heaven with my bootless cries,

And look upon myself and curse my fate,

Asegúrate de separar las cuatro líneas con saltos de línea. A continuación, introduce esto en el intérprete de comandos interactivo:

```
> > > sonnet_file = open(Path.home() / 'sonnet29.txt',  
encoding='UTF-8')
```

```
> > > sonnet_file.readlines()
```

```
["When, in disgrace with fortune and men's eyes,\n", 'I all alone  
beweep
```

```
my outcast state,\n', 'And trouble deaf heaven with my bootless  
cries,\n',
```

```
'And look upon myself and curse my fate,']
```

Observa que, salvo la última línea del archivo, cada uno de los valores de cadena termina con un carácter de salto de línea `\n`. Suele ser más fácil trabajar con una lista de cadenas que con un único valor de cadena grande.

Escribir en archivos

Python permite escribir contenido en un archivo, igual que la función `print()` escribe cadenas en la pantalla. No se puede escribir en un archivo abierto en modo lectura; hay que abrirlo en modo escritura o agregar.

El modo escritura sobrescribirá el archivo existente, algo parecido a reemplazar el valor de una variable por un nuevo valor. Pasa `'w'` como segundo argumento a `open()` para abrir el archivo en modo escritura. Por su parte, el modo agregar añadirá texto al final del archivo existente. Puedes comparar este modo con añadir valores a una lista en una variable en vez de sobrescribir la variable. Pasa `'a'` como segundo argumento a `open()` para abrir el archivo en modo agregar.

Si el nombre de archivo que pasamos a `open()` no existe, tanto el modo escritura como el modo agregar crearán un nuevo archivo en blanco. Después de leer o escribir un archivo, llama al método

close() antes de volver a abrir el archivo. Vamos a unir estos conceptos. Escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > bacon_file = open('bacon.txt', 'w', encoding='UTF-8')
```

```
> > > bacon_file.write('Hello, world!\n')
```

14

```
> > > bacon_file.close()
```

```
> > > bacon_file = open('bacon.txt', 'a', encoding='UTF-8')
```

```
> > > bacon_file.write('Bacon is not a vegetable.')
```

25

```
> > > bacon_file.close()
```

```
> > > bacon_file = open('bacon.txt', encoding='UTF-8')
```

```
> > > content = bacon_file.read()
```

```
> > > bacon_file.close()
```

```
> > > print(content)
```

Hello, world!

Bacon is not a vegetable.

Primero, abrimos bacon.txt en modo escritura. Como no existe todavía, Python lo crea. Llamar a write() con el archivo abierto y pasarle el argumento de cadena 'Hello, world!\n' escribe la cadena en el archivo y devuelve el número de caracteres escritos, incluido el salto de línea. Después, cerramos el archivo.

Para añadir texto al contenido actual del archivo en lugar de reemplazar lo que acabamos de escribir, abrimos el archivo en modo agregar. Escribimos 'Bacon is not a vegetable.' en el archivo y lo cerramos. Por último, para imprimir el contenido del archivo en la pantalla, abrimos el archivo en el modo lectura predeterminado,

llamamos a `read()`, almacenamos el objeto `File` en `content`, cerramos el archivo e imprimimos `content`.

Observa que el método `write()` no agrega automáticamente un salto de línea al final de la cadena como hace la función `print()`. Tendrás que añadir tú ese carácter.

También se puede pasar un objeto `Path` a la función `open()` en lugar del nombre de archivo como cadena.

Usar declaraciones `with`

Con todos los archivos con los que tu programa llame a `open()` hay que llamar a `close()` también, pero es posible que olvides incluir esta función o que el programa se la salte en determinadas circunstancias.

La declaración `with` de Python hace más fácil cerrar archivos automáticamente. Una declaración `with` crea una cosa llamada “gestor de contexto” que Python utiliza para administrar recursos. Estos recursos, como archivos, conexiones de red o segmentos de memoria, a menudo tienen pasos de configuración y destrucción durante los cuales se asigna el recurso y luego se libera para que puedan usarlo otros programas. (No obstante, la mayor parte del tiempo verás declaraciones `with` utilizadas para abrir archivos).

La declaración `with` abre un bloque de código que empieza asignando el recurso y luego lo libera cuando la ejecución del programa sale del bloque, lo cual puede suceder por una declaración `return`, una excepción no manejada o cualquier otra razón. Este es el típico código que lee y escribe el contenido de un archivo:

```
file_obj = open('data.txt', 'w', encoding='utf-8')
```

```
file_obj.write('Hello, world!')
```

```
file_obj.close()
```

```
file_obj = open('data.txt', encoding='utf-8')
```

```
content = file_obj.read()
```

```
file_obj.close()
```

Este es el código equivalente con una declaración with:

```
with open('data.txt', 'w', encoding='UTF-8') as file_obj:
```

```
file_obj.write('Hello, world!')
```

```
with open('data.txt', encoding='UTF-8') as file_obj:
```

```
content = file_obj.read()
```

En el ejemplo de la declaración with, fíjate en que no hay llamadas a close() porque with la llama automáticamente cuando la ejecución del programa sale del bloque. La declaración with sabe hacer eso basándose en el gestor de contexto que obtiene de la función open(). La creación del gestor de contexto queda fuera del ámbito de este libro, pero puedes aprender más sobre el tema con la documentación en línea en <https://docs.python.org/3/reference/datamodel.html#context-managers> o el libro *Serious Python* de Julien Danjou (No Starch Press, 2018).

Guardar variables con el módulo shelve

Podemos guardar variables de nuestros programas de Python en archivos binarios usando el módulo shelve. Esto permite que el programa restablezca esos datos a las variables la próxima vez que se ejecute. Podrías usar esta técnica para añadir a tu programa funciones de guardar y abrir; por ejemplo, si ejecutases un programa e introdujeses ajustes de configuración, podrías guardar esos ajustes en un archivo estante y hacer que el programa cargue esos ajustes la próxima vez que se ejecute. Para practicar con shelve, introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > import shelve
```

```
> > > shelf_file = shelve.open('mydata')
```

```
> > > shelf_file['cats'] = ['Zophie', 'Pooka', 'Simon']
```

```
> > > shelf_file.close()
```

Para leer y escribir datos con el módulo `shelve`, primero hay que importarlo. A continuación, llamamos a `shelve.open()` y le pasamos un nombre de archivo y guardamos el valor estante devuelto en una variable. Podemos hacer cambios en este valor como si fuese un diccionario. Al terminar, llamamos a `close()` con el valor estante. Aquí, este valor está guardado en `shelf_file`. Creamos una lista `cats` y escribimos `shelf_file['cats'] = ['Zophie', 'Pooka', 'Simon']` para almacenar la lista en `shelf_file` como valor asociado con la clave `'cats'` (como en un diccionario). Luego, llamamos a `close()` con `shelf_file`.

Tras ejecutar el código anterior en Windows, deberías ver tres nuevos archivos en el directorio de trabajo actual: `mydata.bak`, `mydata.dat` y `mydata.dir`. En macOS, deberías ver solo un archivo `mydata.db` y en Linux, solo `mydata`. Los archivos binarios contienen los datos almacenados en el estante. El formato de estos archivos binarios no importa; solo necesitas saber lo que hace el módulo `shelve`, no cómo lo hace. Este módulo te libera de la preocupación por cómo guardar los datos de tu programa en un archivo.

Tus programas pueden usar el módulo `shelve` para reabrir y recuperar posteriormente datos de estos archivos estantes. Los valores estantes no necesitan abrirse en modo lectura o escritura; permiten leer y escribir cuando se abren. Escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > shelf_file = shelve.open('mydata')
```

```
> > > type(shelf_file)
```

```
<class 'shelve.DbfilenameShelf'>
```

```
> > > shelf_file['cats']
```

```
['Zophie', 'Pooka', 'Simon']
```

```
> > > shelf_file.close()
```

Aquí, abrimos los archivos estantes para comprobar que han guardado bien los datos. Introducir `shelf_file['cats']` devuelve la misma lista de antes. Ahora que sabemos que el archivo tiene bien la lista, llamamos a `close()`.

Igual que los diccionarios, los valores estante tienen métodos `keys()` y `values()` que devolverán valores de tipo lista con las claves y los valores del estante. Dado que estos valores de retorno no son listas de verdad, deberíamos pasarlas a la función `list()` para obtenerlas en forma de lista. Escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > shelf_file = shelve.open('mydata')
```

```
> > > list(shelf_file.keys())
```

```
['cats']
```

```
> > > list(shelf_file.values())
```

```
[['Zophie', 'Pooka', 'Simon']]
```

```
> > > shelf_file.close()
```

El texto simple es útil para crear archivos que leerás en el Bloc de notas o TextEdit, pero, si quieres guardar datos de tus programas de Python, usa el módulo `shelve`.

Proyecto 4: Generar archivos de cuestionarios aleatorios

Imagina que eres profesor de geografía, tienes 35 alumnos en clase y quieres hacerles un examen sorpresa de las capitales de EE. UU. Desafortunadamente, tienes unos cuantos revoltosos en clase y no puedes confiar en que no copien. Quieres aleatorizar el orden de las preguntas para que cada cuestionario sea único, de modo que sea imposible que unos copien las respuestas de otros. Naturalmente, hacer esto a mano sería una tarea larga y tediosa. Por suerte, sabes un poco de Python.

Esto es lo que hace el programa:

- Crear 35 cuestionarios diferentes.
- Crear 50 preguntas de tipo test para cada cuestionario, en orden aleatorio.
- Proporcionar la respuesta correcta y tres respuestas incorrectas aleatorias para cada pregunta, en orden aleatorio.
- Escribir los cuestionarios en 35 archivos de texto.
- Escribir las claves de respuesta en 35 archivos de texto.

Esto significa que el código tendrá que hacer lo siguiente:

- Guardar los estados y sus capitales en un directorio.
- Llamar a `open()`, `write()` y `close()` para los archivos de texto de las preguntas y las respuestas correctas.
- Usar `random.shuffle()` para aleatorizar el orden de las preguntas y las opciones de respuesta.

Vamos allá.

Paso 1: guardar los datos del cuestionario en un diccionario

El primer paso consiste en crear un *script* que sirva como esqueleto y rellenarlo con los datos del cuestionario. Crea un archivo llamado `randomQuizGenerator.py` y escribe lo siguiente:

```
# randomQuizGenerator.py – Crea cuestionarios con preguntas y respuestas en
```

```
# orden aleatorio, junto con la clave de respuestas
```

```
1 import random
```

```
# Los datos del cuestionario. Las claves son los estados y los valores sus capitales.
```

```
2 capitals = {'Alabama': 'Montgomery', 'Alaska': 'Juneau',  
'Arizona':
```

'Phoenix', 'Arkansas': 'Little Rock', 'California': 'Sacramento',
'Colorado':

'Denver', 'Connecticut': 'Hartford', 'Delaware': 'Dover', 'Florida':

'Tallahassee', 'Georgia': 'Atlanta', 'Hawaii': 'Honolulu', 'Idaho':
'Boise',

'Illinois': 'Springfield', 'Indiana': 'Indianapolis', 'Iowa': 'Des Moines',

'Kansas': 'Topeka', 'Kentucky': 'Frankfort', 'Louisiana': 'Baton
Rouge',

'Maine': 'Augusta', 'Maryland': 'Annapolis', 'Massachusetts': 'Boston',

'Michigan': 'Lansing', 'Minnesota': 'Saint Paul', 'Mississippi':
'Jackson',

'Missouri': 'Jefferson City', 'Montana': 'Helena', 'Nebraska':
'Lincoln',

'Nevada': 'Carson City', 'New Hampshire': 'Concord', 'New Jersey':
'Trenton',

'New Mexico': 'Santa Fe', 'New York': 'Albany', 'North Carolina':
'Raleigh',

'North Dakota': 'Bismarck', 'Ohio': 'Columbus', 'Oklahoma':
'Oklahoma City',

'Oregon': 'Salem', 'Pennsylvania': 'Harrisburg', 'Rhode Island':
'Providence',

'South Carolina': 'Columbia', 'South Dakota': 'Pierre', 'Tennessee':

'Nashville', 'Texas': 'Austin', 'Utah': 'Salt Lake City', 'Vermont':

'Montpelier', 'Virginia': 'Richmond', 'Washington': 'Olympia', 'West

Virginia': 'Charleston', 'Wisconsin': 'Madison', 'Wyoming':
'Cheyenne'}

Genera 35 archivos de cuestionario.

3 for quiz_num in range(35):

PENDIENTE: Crear los archivos de preguntas y respuestas correctas.

PENDIENTE: Escribir el encabezado del cuestionario.

PENDIENTE: Revolver el orden de los estados.

PENDIENTE: Pasar en bucle por los 50 estados, haciendo una pregunta para cada uno.

Como este programa ordenará aleatoriamente las preguntas y las respuestas, hay que importar el módulo random 1 para poder usar estas funciones. La variable capitals 2 contiene un diccionario con los estados como claves y sus capitales como valores. Y, como queremos crear 35 cuestionarios, el código que realmente genera los archivos para los cuestionarios y las respuestas (de momento, marcado como PENDIENTE en los comentarios) irá dentro de un bucle for que itere 35 veces 3. (Este número se puede cambiar para generar cualquier cantidad de archivos de cuestionario).

Paso 2: crear el archivo de cuestionario

Ha llegado el momento de empezar a rellenar todo lo PENDIENTE.

El código del bucle se repetirá 35 veces, una por cada cuestionario, así que solo tenemos que preocuparnos por un cuestionario cada vez dentro del bucle. Primero, crearemos el archivo de cuestionario real. Necesita un nombre único y algún tipo de encabezado estándar, con espacio para que los estudiantes pongan el nombre, la fecha y el trimestre. A continuación, hay que obtener una lista de estados en orden aleatorio, que usaremos después para crear las preguntas y respuestas.

Añade estas líneas de código a randomQuizGenerator.py:

randomQuizGenerator.py - Crea cuestionarios con preguntas y respuestas en

orden aleatorio, junto con la clave de respuestas

--fragmento omitido--

```
# Genera 35 archivos de cuestionario.
```

```
for quiz_num in range(35):
```

```
# Crea los archivos de preguntas y respuestas correctas.
```

```
quiz_file = open(f'capitalsquiz{quiz_num + 1}.txt', 'w',  
encoding='UTF-8') 1
```

```
answer_file = open(f'capitalsquiz_answers{quiz_num + 1}.txt',  
'w', encoding='UTF-8') 2
```

```
# Escribe el encabezado del cuestionario.
```

```
quiz_file.write('Name:\n\nDate:\n\nPeriod:\n\n') 3
```

```
quiz_file.write((' ' * 20) + f'State Capitals Quiz  
(Form{quiz_num + 1})')
```

```
quiz_file.write('\n\n')
```

```
# Revuelve el orden de los estados.
```

```
states = list(capitals.keys())
```

```
random.shuffle(states) 4
```

```
# PENDIENTE: Pasar en bucle por los 50 estados, haciendo una  
pregunta para cada uno.
```

Los cuestionarios utilizarán los nombres de archivo capitalsquiz<N>.txt, donde <N> es un número único procedente de quiz_num, el contador del bucle. Guardaremos las respuestas correctas para capitalsquiz<N>.txt en archivos llamados capitalsquiz_answers<N>.txt. Con cada iteración del bucle, el código sustituirá los marcadores de posición {quiz_num + 1} en estos nombres de archivo por un número único. Por ejemplo, nombra el primer cuestionario y las respuestas correspondientes key capitalsquiz1.txt y capitalsquiz_answers1.txt. Creamos estos archivos con llamadas a la función open() en 1 y 2, pasando 'w'

como segundo argumento para abrirlos en modo escritura.

La declaración `write()` en 3 crea un encabezado para que lo rellene el estudiante. Por último, generamos una lista aleatoria de los Estados Unidos con ayuda de la función `random.shuffle()` 4, que reorganiza al azar los valores en cualquier lista que se le pase.

Paso 3: crear las opciones de respuesta

Ahora necesitamos generar opciones de respuesta de A a D para cada pregunta con otro bucle `for`. Después, un tercer bucle `for` anidado las escribirá en los archivos. Añade esto al código:

```
# randomQuizGenerator.py - Crea cuestionarios con preguntas y respuestas en
```

```
# orden aleatorio, junto con la clave de respuestas
```

```
--fragmento omitido--
```

```
# Pasa en bucle por los 50 estados, haciendo una pregunta para cada uno.
```

```
for num in range(50):
```

```
# Obtiene respuestas correctas e incorrectas.
```

```
correct_answer = capitals[states[num]]
```

```
wrong_answers = list(capitals.values())
```

```
del wrong_answers[wrong_answers.index(correct_answer)]
```

```
wrong_answers = random.sample(wrong_answers, 3)
```

```
answer_options = wrong_answers + [correct_answer]
```

```
random.shuffle(answer_options)
```

```
# PENDIENTE: Escribir la pregunta y las opciones de respuesta en el archivo
```

del cuestionario.

PENDIENTE: Escribir las respuestas correctas en un archivo.

La respuesta correcta es fácil de crear; ya está almacenada como valor en el diccionario capitals. Este bucle iterará por los estados en la lista states revuelta, buscará cada estado en capitals y guardará la capital correspondiente al estado en correct_answer.

Crear la lista de posibles respuestas incorrectas es más difícil. Podemos obtenerlas duplicando los valores del diccionario capitals, eliminando la respuesta correcta y seleccionando tres valores aleatorios de esta lista. La función random.sample() facilita esta selección. Su primer argumento es la lista de la que queremos seleccionar y el segundo, el número de valores que seleccionar. La lista completa de opciones de respuesta combina estas tres opciones incorrectas con las respuestas correctas. Por último, aleatorizamos las respuestas para que la correcta no sea siempre la D.

Paso 4: escribir contenido en los archivos

Ya solo nos queda escribir la pregunta en el archivo del cuestionario y la respuesta en el de respuestas. Añade este código al programa:

```
# randomQuizGenerator.py - Crea cuestionarios con preguntas y respuestas en
```

```
# orden aleatorio, junto con la clave de respuestas
```

```
--fragmento omitido--
```

```
# Pasa en bucle por los 50 estados, haciendo una pregunta para cada uno.
```

```
for num in range(50):
```

```
--fragmento omitido--
```

```
# Escribe la pregunta y las opciones de respuesta en el archivo del cuestionario.
```

```
quiz_file.write(f'{num + 1}. Capital of {states[num]}:\n')
```

```

1 for i in range(4):
2 quiz_file.write(f" {'ABCD' [i] }. { answer_options [i] }\n")
quiz_file.write('\n')

# Escribe la respuesta correcta en un archivo.

3 answer_file.write(f"{num + 1}.
{'ABCD' [answer_options.index(correct_answer)] }")

quiz_file.close()

answer_file.close()

```

Un bucle for itera por enteros de 0 a 3 para escribir las opciones de respuesta en la lista `answer_options` del archivo 1. La expresión `'ABCD'[i]` en 2 trata la cadena `'ABCD'` como una matriz y se evaluará a `'A'`, `'B'`, `'C'` y `'D'` en cada iteración por el bucle.

En la última línea del bucle, la expresión `answer_options.index(correct_answer)` 3 buscará el índice entero de la respuesta correcta en las opciones de respuesta ordenadas aleatoriamente, haciendo que la letra de la respuesta correcta se escriba en el archivo de respuestas correctas.

Una vez ejecutado el programa, el archivo `capitalsquiz1.txt` debería ser algo parecido a esto. Obviamente, las preguntas y opciones de respuesta dependerán de cada llamada a `random.shuffle()`:

Name:

Date:

Period:

State Capitals Quiz (Form 1)

1. What is the capital of West Virginia?

A. Hartford

B. Santa Fe

C. Harrisburg

D. Charleston

2. What is the capital of Colorado?

A. Raleigh

B. Harrisburg

C. Denver

D. Lincoln

--fragmento omitido--

El archivo capitalsquiz_answers1.txt sería así:

1. D

2. C

--fragmento omitido--

Ordenar aleatoriamente las preguntas y las respuestas correctas a mano nos llevaría horas, pero, con un poco de conocimiento de programación, podemos automatizar esta aburrida tarea no solo para el examen de capitales, sino para cualquier examen tipo test.

Resumen

Los sistemas operativos organizan los archivos en carpetas (también llamadas directorios) y utilizan rutas para describir sus ubicaciones. Cada programa que se ejecuta en un ordenador tiene un directorio actual de trabajo, lo que nos permite especificar rutas de archivo relativas a la ubicación actual, en lugar de introducir la ruta completa (o absoluta). Los módulos `pathlib` y `os.path` cuentan con varias funciones para manipular rutas de archivo.

Tus programas también pueden interactuar directamente con el contenido de archivos de texto. La función `open()` puede abrir los archivos para leer su contenido como una cadena grande (con el

método `read()` o como una lista de cadenas (con el método `readlines()`). La función `open()` también puede abrir archivos en modo escritura o agregar para crear nuevos archivos de texto o añadir algo a otros ya existentes, respectivamente.

En capítulos anteriores, hemos usado el portapapeles como medio para llevar grandes cantidades de texto a un programa en lugar de escribirlo directamente. Ahora, puedes hacer que tus programas lean archivos del disco duro, lo cual es una mejora importante, ya que los archivos son mucho menos volátiles que el portapapeles.

En el próximo capítulo, aprenderás a manejar los archivos en sí, copiándolos, borrándolos, renombrándolos, moviéndolos, etc.

Preguntas prácticas

1. ¿A qué es relativa una ruta relativa?
2. ¿Con qué empieza una dirección absoluta?
3. ¿A qué se evalúa `Path('C:/Users')` / `'A'` en Windows?
4. ¿A qué se evalúa `'C:/Users'` / `'A'` en Windows?
5. ¿Qué hacen las funciones `os.getcwd()` y `os.chdir()`?
6. ¿Qué son las carpetas `.` y `..`?
7. En `C:\bacon\eggs\spam.txt`, ¿qué parte es el nombre de la carpeta y qué parte es la base del nombre?
8. ¿Qué tres argumentos de “modo” se pueden pasar a la función `open()` para archivos de texto simple?
9. ¿Qué pasa si se abre un archivo existente en modo escritura?
10. ¿Cuál es la diferencia entre los métodos `read()` y `readlines()`?
11. ¿A qué estructura de datos se parece un valor estante?

Programas de práctica

Para practicar, diseña y escribe los siguientes programas.

Mad Libs

Crea un programa de *Mad Libs* que lea en archivos de texto y permita al usuario añadir su propio texto en cualquier parte donde aparezcan las palabras ADJECTIVE, NOUN, ADVERB o VERB en el archivo de texto. Por ejemplo, el archivo de texto puede ser así:

The ADJECTIVE panda walked to the NOUN and then VERB. A nearby NOUN was

unaffected by these events.

El programa encontraría estos casos y pediría al usuario que los sustituya:

Enter an adjective:

silly

Enter a noun:

chandelier

Enter a verb:

screamed

Enter a noun:

pickup truck

A continuación, crearía el siguiente texto:

The silly panda walked to the chandelier and then screamed. A nearby

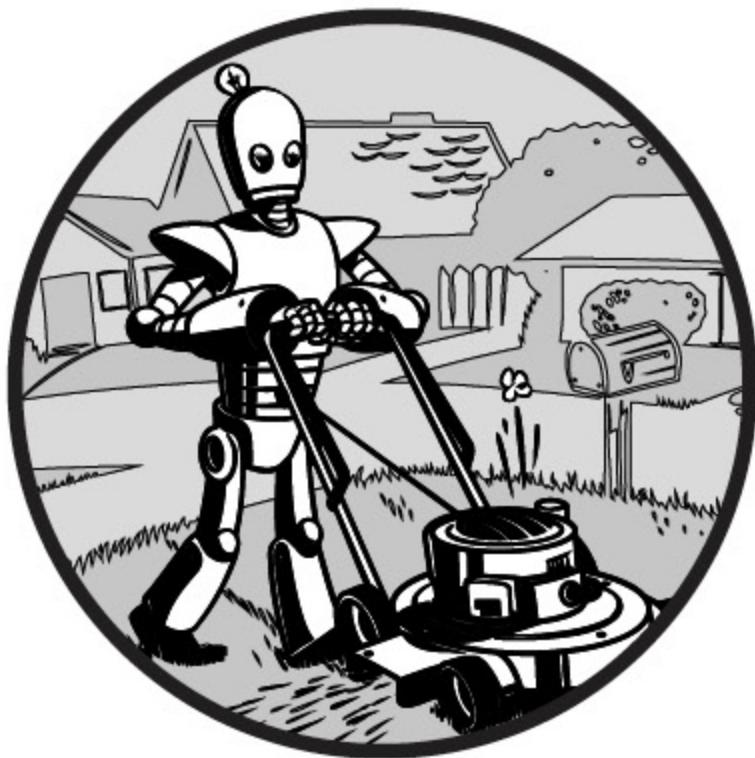
pickup truck was unaffected by these events.

El programa debería imprimir los resultados en la pantalla, además de guardarlos en nuevo archivo de texto.

Buscar expresiones regulares

Escribe un programa que abra todos los archivos .txt de una carpeta para buscar cualquier línea que coincida con una expresión regular proporcionada por el usuario y después imprima los resultados en la pantalla.

Organizar archivos



Además de crear nuevos archivos y escribir en ellos, tus programas pueden organizar archivos que ya tenías en el disco duro. Tal vez hayas tenido la experiencia de pasar por decenas, cientos o incluso miles de archivos y copiarlos, renombrarlos, moverlos o comprimirlos todos a mano. O piensa en tareas como estas:

- Hacer copias de todos los PDF (solo los PDF) de todas las subcarpetas de una carpeta.
- Quitar los ceros a la izquierda de todos los archivos de una carpeta que contiene cientos de documentos llamados spam001.txt,

spam002.txt, spam003.txt, etc.

- Comprimir el contenido de varias carpetas en un archivo ZIP (que podría servir como sistema de copia de seguridad sencillo).

Estas pesadeces están pidiendo a gritos ser automatizadas con Python. Programando tu ordenador para que haga estas tareas puedes convertirlo en un secretario rápido que nunca se equivoca.

Aunque Windows usa barras invertidas (\) para separar las carpetas en una ruta de archivo, el código Python de este capítulo usará barras oblicuas (/) porque funcionan en todos los sistemas operativos.

El módulo shutil

El módulo `shutil` tiene funciones que nos permiten copiar, mover, renombrar y borrar archivos en programas de Python. (El nombre del módulo es la abreviatura de *shell utilities*, donde *shell* se refiere a una línea de comandos). Para usar las funciones de `shutil`, primero hay que ejecutar `import shutil`.

Para crear un archivo y una carpeta de ejemplo con los que trabajar, ejecuta el siguiente código antes de los ejemplos del intérprete de comandos interactivo de este capítulo:

```
> > > from pathlib import Path
> > > h = Path.home()
> > > (h / 'spam').mkdir(exist_ok=True)
> > > with open(h / 'spam/file1.txt', 'w', encoding='utf-8') as
file:
... file.write('Hello')
...
```

Esto creará una carpeta llamada `spam` con un archivo de texto `file1.txt`. Los ejemplos de este capítulo copiarán, moverán, renombrarán y borrarán este archivo y la carpeta. Todas las

funciones de `shutil` pueden tomar argumentos de ruta de archivo que pueden ser cadenas u objetos `Path`.

Copiar archivos y carpetas

El módulo `shutil` ofrece funciones para copiar archivos, además de carpetas enteras. Llamar a `shutil.copy(origen, destino)` copiará el archivo de la ruta de origen a la ruta de destino. Ambas rutas pueden ser cadenas y objetos `Path`. Si el destino es un nombre de archivo, se usará como nuevo nombre para el archivo copiado. Si el destino es una carpeta, el archivo se copiará en esa carpeta con su nombre original. Esta función devuelve la ruta del archivo copiado.

Introduce lo siguiente en el intérprete de comandos interactivo para ver cómo funciona `shutil.copy()`:

```
> > > import shutil
```

```
> > > from pathlib import Path
```

```
> > > h = Path.home()
```

```
1 > > > shutil.copy(h / 'spam/file1.txt', h)
```

```
'C:\\Users\\Al\\file1.txt'
```

```
2 > > > shutil.copy(h / 'spam/file1.txt', h / 'spam/file2.txt')
```

```
WindowsPath('C:/Users/Al/spam/file2.txt')
```

La primera llamada a `shutil.copy()` copia el archivo que hay en `C:\Users\Al\spam\file1.txt` en la carpeta de inicio `C:\Users\Al`. El valor de retorno es la ruta al archivo recién copiado. Observa que, como hemos especificado una carpeta como destino 1, el nuevo archivo copiado conservará el nombre del archivo original, `file1.txt`. La segunda llamada a `shutil.copy()` 2 copia el archivo de `C:\Users\Al\spam\file1.txt` a la carpeta `C:\Users\Al\spam` pero pone al archivo copiado el nombre `file2.txt`.

Mientras que `shutil.copy()` copia un solo archivo, llamar a `shutil.copytree(origen, destino)` copia la carpeta en la ruta de origen,

junto con todos sus archivos y subcarpetas, a la ruta de destino. La función devuelve la ruta de la carpeta copiada.

Escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > import shutil  
> > > from pathlib import Path  
> > > h = Path.home()  
> > > shutil.copytree(h / 'spam', h / 'spam_backup')
```

```
WindowsPath('C:/Users/Al/spam_backup')
```

La llamada a `shutil.copytree()` puede crear una nueva carpeta llamada `spam_backup` con el mismo contenido que la carpeta `spam` original. Ya tienes una copia de seguridad de tu valiosa carpeta `spam`.

Mover y renombrar archivos y carpetas

Llamar a `shutil.move(origen, destino)` moverá el archivo o carpeta de la ruta original a la ruta de destino y devolverá una cadena de la ruta absoluta a la nueva ubicación.

Si la ruta de destino apunta a una carpeta, el archivo original se mueve al destino y mantiene su nombre actual. Por ejemplo, introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > import shutil  
> > > from pathlib import Path  
> > > h = Path.home()  
> > > (h / 'spam2').mkdir()  
> > > shutil.move(h / 'spam/file1.txt', h / 'spam2')
```

```
'C:\\Users\\Al\\spam2\\file1.txt'
```

Después de crear la carpeta spam2 en la carpeta de inicio, esta llamada a `shutil.move()` dice: “mueve C:\Users\Al\spam\file1.txt a la carpeta C:\Users\Al\spam2”. Si ya hubiese habido un archivo `file1.txt` en C:\Users\Al\spam2, Python lo habría sustituido.

Si la ruta de destino no es una carpeta existente, `shutil.move()` usará esta ruta para renombrar el archivo. En el siguiente ejemplo, se mueve y renombra el archivo de origen:

```
> > > shutil.move(h / 'spam/file1.txt', h / 'spam2/  
new_name.txt')
```

```
'C:\\Users\\Al\\spam2\\new_name.txt'
```

Esta línea dice: “mueve C:\Users\Al\spam\file1.txt a la carpeta C:\Users\Al\spam2 y, mientras estás ahí, renombra ese archivo `file1.txt` como `new_name.txt`”.

Borrar archivos y carpetas de manera permanente

Podemos borrar un solo archivo o una sola carpeta vacía con funciones del módulo `os`, pero, para borrar una carpeta con todo su contenido, hay que usar el módulo `shutil`:

- Llamar a `shutil.rmtree(ruta)` eliminará (es decir, borrará) todo el árbol de carpetas de la ruta, incluidos todos los archivos y carpetas que haya dentro.
- Llamar a `os.unlink(ruta)` eliminará el único archivo de la ruta.
- Llamar a `os.rmdir(ruta)` borrará la carpeta de la ruta. Esta carpeta debe estar vacía.

Ten cuidado cuando utilices estas funciones en tus programas. A menudo, es mejor ejecutar el programa con estas llamadas anuladas con comentarios y llamadas a `print()` para ver los archivos que se borrarían. Esto se conoce como simulacro. Aquí tenemos un programa de Python que pretendía borrar los archivos con la extensión `.txt`, pero tiene una errata que hace que borre los `.rxt` en su lugar:

```
import os

from pathlib import Path

for filename in Path.home().glob('*.*txt'):

    os.unlink(filename)
```

Si tuviésemos archivos importantes que acabasen en `.txt`, habrían sido borrados por accidente de forma permanente. Podríamos haber ejecutado el programa primero así:

```
import os

from pathlib import Path

for filename in Path.home().glob('*.*txt'):

    #os.unlink(filename)

    print('Deleting', filename)
```

La llamada a `os.unlink()` está comentada, así que Python la ignora. En su lugar, imprimimos el nombre de archivo o el archivo que se habría borrado. Ejecutar esta versión del programa primero nos muestra que hemos dicho sin querer al programa que borre los archivos `.txt` en vez de los `.txt`.

También deberías hacer simulacros para programas que copien, renombren o muevan archivos. Por último, puede ser buena idea crear una copia de seguridad de toda la carpeta que vaya a tocar el programa, por si necesitas restaurar los archivos originales en algún momento. Cuando estés seguro de que el programa funciona como esperabas, borra la línea `print(filename)` y quita el comentario de `os.unlink(filename)`. Luego ejecuta el programa de nuevo para borrar de verdad los archivos.

Enviar a la Papelera de reciclaje

La función integrada de Python `shutil.rmtree()` borra de manera irreversible archivos y carpetas. Esto hace que sea peligroso usar esta función, porque un fallo podría eliminar archivos que no

queríamos borrar. Una forma mucho mejor de eliminar archivos y carpetas es con el módulo de terceros `send2trash`. (Consulta el apéndice A para una explicación más detallada de cómo instalar paquetes de terceros). Usar la función `send2trash()` del módulo `send2trash` es mucho más seguro que emplear las funciones de eliminación propias de Python, ya que envía los archivos y carpetas a la Papelera de reciclaje de tu ordenador en vez de borrarlos permanentemente. Si un fallo en el programa borra con `send2trash` algo que no querías eliminar, puedes recuperarlo de la papelera.

Cuando hayas instalado `send2trash`, escribe lo siguiente en el intérprete de comandos interactivo para enviar el archivo `file1.txt` a la Papelera de reciclaje:

```
> > > import send2trash  
  
> > > send2trash.send2trash('file1.txt')
```

En general, deberías usar la función `send2trash.send2trash()` para borrar archivos y carpetas. Pero, aunque enviar archivos a la papelera permite recuperarlos después, no liberará espacio en el disco duro como hace borrarlos permanentemente. Ten en cuenta que la función `send2trash()` solo puede mandar archivos a la papelera, no puede sacarlos de ella.

Recorrer un árbol de directorios

Si quieres una lista de todos los archivos y subcarpetas que hay en una carpeta, llama a la función `os.listdir()` y pásale un nombre de carpeta:

```
> > > import os  
  
> > > os.listdir(r'C:\Users\Al')  
  
['.anaconda', '.android', '.cache', '.dotnet', '.eclipse', '.gitconfig',  
--fragmento omitido--  
['_pycache_']
```

También puedes obtener una lista de objetos Path en una carpeta llamando al método `iterdir()`:

```
> > > from pathlib import Path
```

```
> > > home = Path.home()
```

```
> > > list(home.iterdir())
```

```
[WindowsPath('C:/Users/Al/.anaconda'), WindowsPath('C:/Users/Al/.android'),
```

```
WindowsPath('C:/Users/Al/.cache'),
```

```
--fragmento omitido--
```

```
WindowsPath('C:/Users/Al/_pycache_')]
```

Supongamos que queremos renombrar todos los archivos de una carpeta y también todos los que haya en todas las subcarpetas de esa carpeta. Es decir, queremos recorrer el árbol de directorios, accediendo a todos los archivos. Escribir un programa que pueda hacer eso sería difícil, pero, por suerte, Python cuenta con la función `os.walk()` para manejar este proceso por nosotros.

Vamos a crear una serie de carpetas y archivos ejecutando este código en el intérprete de comandos interactivo:

```
> > > from pathlib import Path
```

```
> > > h = Path.home()
```

```
> > > (h / 'spam').mkdir(exist_ok=True)
```

```
> > > (h / 'spam/eggs').mkdir(exist_ok=True)
```

```
> > > (h / 'spam/eggs2').mkdir(exist_ok=True)
```

```
> > > (h / 'spam/eggs/bacon').mkdir(exist_ok=True)
```

```
> > > for f in ['spam/file1.txt', 'spam/eggs/file2.txt', 'spam/eggs/file3.txt',
```

`‘spam/eggs/bacon/file4.txt’]` :

`. with open(h / f, ‘w’, encoding=‘utf-8’) as file:`

`... file.write(‘Hello’)`

`.`

`> > > # En este punto, las carpetas y archivos ya existen.`

Este código creará los siguientes directorios y archivos en tu carpeta inicio:

- La carpeta spam.
- El archivo spam/file1.txt.
- La carpeta spam/eggs.
- El archivo spam/eggs/file2.txt.
- El archivo spam/eggs/file3.txt.
- La carpeta spam/eggs2.
- La carpeta spam/eggs/bacon.
- El archivo spam/eggs/bacon/file4.txt.

Aquí tenemos un programa de ejemplo que utiliza la función `os.walk()` con este árbol de carpetas y renombra cada archivo para ponerlo en mayúsculas:

```
import os, shutil
```

```
from pathlib import Path
```

```
h = Path.home()
```

```
for folder_name, subfolders, filenames in os.walk(h / ‘spam’):
```

```
print(‘The current folder is ‘ + folder_name)
```


for subfolder in subfolders:

```
print("SUBFOLDER OF " + folder_name + ": " + subfolder)
```

for filename in filenames:

```
print("FILE INSIDE " + folder_name + ": " + filename)
```

Renombra el archivo en mayúsculas:

```
p = Path(folder_name)
```

```
shutil.move(p / filename, p / filename.upper())
```

```
print("")
```

La función `os.walk()` recibe un único valor de cadena: la ruta de un archivo. Podemos usar `os.walk()` en un bucle `for` para recorrer un árbol de directorios, igual que podemos usar la función `range()` para recorrer un rango de números.

A diferencia de `range()`, la función `os.walk()` devolverá tres valores en cada iteración por el bucle:

- Una cadena del nombre de la carpeta actual.
- Una lista de cadenas de las subcarpetas de la carpeta actual.
- Una lista de cadenas de los archivos en la carpeta actual.

La carpeta actual aquí se refiere a la carpeta a la que se ha accedido en la actual iteración del bucle `for`. La función `os.walk()` no cambia el directorio de trabajo actual del programa. Igual que puedes elegir el nombre de variable `i` en el código `for i in range(10):`, puedes elegir los nombres de las variables para los tres valores enumerados antes. Yo siempre uso nombres descriptivos, como `folder_name`, `subfolders` y `filenames`.

Cuando ejecuté este programa en mi ordenador, produjo la siguiente salida:

The current folder is C:\Users\Al\spam

SUBFOLDER OF C:\Users\Al\spam: eggs

SUBFOLDER OF C:\Users\Al\spam: eggs2

FILE INSIDE C:\Users\Al\spam: file1.txt

The current folder is C:\Users\Al\spam\eggs

SUBFOLDER OF C:\Users\Al\spam\eggs: bacon

FILE INSIDE C:\Users\Al\spam\eggs: file2.txt

FILE INSIDE C:\Users\Al\spam\eggs: file3.txt

The current folder is C:\Users\Al\spam\eggs\bacon

FILE INSIDE C:\Users\Al\spam\eggs\bacon: file4.txt

The current folder is C:\Users\Al\spam\eggs2

Como `os.walk()` devuelve listas de cadenas y variables de nombre de archivo, podemos usar los valores de retorno en sus propios bucles `for`. A saber, podemos pasar la carpeta y el nombre de archivo a funciones como `shutil.move()`, como en el ejemplo.

Comprimir archivos con el módulo `zipfile`

Es posible que estés familiarizado con los archivos ZIP (con la extensión `.zip`), que pueden acoger el contenido comprimido de otros muchos archivos. Comprimir un archivo reduce su tamaño, lo cual es útil cuando se transfiere por Internet. Y, dado que un archivo ZIP también puede contener varios archivos y carpetas, es una forma útil de empaquetar varios archivos en uno. Este archivo puede, por ejemplo, adjuntarse a un correo electrónico. Tus programas de Python pueden crear archivos ZIP y extraer contenido de ellos con el módulo `zipfile`.

Crear y añadir a archivos ZIP

Para crear tus propios archivos comprimidos ZIP, debes abrir el objeto `ZipFile` en modo escritura pasándole `'w'` como segundo

argumento. (Fíjate en la Z y la F mayúsculas en el nombre del objeto, que lo diferencia del nombre del módulo zipfile). Este proceso es similar a abrir un archivo de texto en modo escritura pasando 'w' a la función open(). Para el nombre de archivo, puedes pasar una cadena o un objeto Path. Cuando pasas una ruta al método write() de un objeto ZipFile, Python comprime el archivo de esa ruta y lo añade al ZIP. El primer argumento de write() es una cadena del nombre de archivo que añadir. El segundo es el parámetro del tipo de compresión, que dice al ordenador qué algoritmo debería emplear para comprimir los archivos; siempre puedes establecer este valor en zipfile.ZIP_DEFLATED para especificar el algoritmo deflación, que funciona bien con todo tipo de datos. Si no pasas este valor, el método write() añade al archivo al ZIP con su tamaño normal sin comprimir. Escribe esto en el intérprete de comandos interactivo:

```
> > > import zipfile

> > > with open('file1.txt', 'w', encoding='utf-8') as file_obj:

. file_obj.write('Hello' * 10000)

.

> > > with zipfile.ZipFile('example.zip', 'w') as example_zip:

. example_zip.write('file1.txt',
compress_type=zipfile.ZIP_DEFLATED,
compresslevel=9)
```

Este código crea un archivo de texto llamado file1.txt y escribe en él la cadena de 50.000 caracteres 'Hello' * 10000 (unos 49KB). Después, crea un archivo ZIP llamado example.zip que tiene el contenido comprimido de file1.txt (unos 213 bytes; los datos muy repetitivos son también muy comprimibles). El argumento de palabra clave compresslevel (incorporado en Python 3.7 y posterior) puede ser cualquier valor entre 0 y 9, siendo 9 el nivel más lento, pero más comprimido. Si no se especifica este argumento, el nivel predeterminado es 6.

La función `zipfile.ZipFile()` abre un archivo ZIP en una declaración `with`, de forma similar a como la función `open()` abre archivos. Esto garantiza que se llame automáticamente al método `close()` cuando la ejecución sale del bloque de la declaración `with`.

Ten en cuenta que, igual que, al escribir archivos, el modo escritura borrará todo el contenido existente en un archivo ZIP. Si solo quieres añadir archivos a un ZIP, pasa 'a' como segundo argumento a `zipfile.ZipFile()` para abrir el ZIP en modo agregar.

Leer archivos ZIP

Para leer el contenido de un archivo ZIP, hay que crear un objeto `ZipFile` llamando a la función `zipfile.ZipFile()` y pasándole el nombre del archivo ZIP. Observa que `zipfile` es el nombre del módulo de Python, y `ZipFile()` el de la función.

Para ver un ejemplo, introduce esto en el intérprete de comandos interactivo:

```
> > > import zipfile

> > > example_zip = zipfile.ZipFile('example.zip')

> > > example_zip.namelist()

['file1.txt']

> > > file1_info = example_zip.getinfo('file1.txt')

> > > file1_info.file_size

50000

> > > file1_info.compress_size

97

1 > > > f'Compressed file is {round(file1_info.file_size /
file1_info
.compress_size, 2)}x smaller!'
```

‘Compressed file is 515.46x smaller!’

```
> > > example_zip.close()
```

Un objeto ZipFile tiene un método `namelist()` que devuelve una lista de cadenas para todos los archivos y carpetas contenidos en el archivo ZIP. Estas cadenas se pueden pasar al método `getinfo()` de ZipFile para devolver un objeto ZipInfo para ese archivo en particular. Los objetos ZipInfo tienen sus propios atributos, como `file_size` y `compress_size`, que contienen enteros que representan el tamaño del archivo original y el del archivo comprimido, respectivamente, en bytes. Un objeto ZipFile representa un archivo entero y un objeto ZipInfo contiene información sobre un archivo de ese archivo.

El comando en 1 calcula la eficiencia con la que se comprime `example.zip` dividiendo el tamaño de archivo original entre el tamaño comprimido, e imprime esta información.

Extraer de archivos ZIP

El método `extractall()` para objetos ZipFile extrae todos los archivos y carpetas de un archivo ZIP en el directorio de trabajo actual. Crea un archivo ZIP llamado `example.zip` siguiendo las instrucciones del apartado “Crear y añadir a archivos ZIP” y, a continuación, introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > import zipfile
```

```
> > > example_zip = zipfile.ZipFile('example.zip')
```

```
1 > > > example_zip.extractall()
```

```
> > > example_zip.close()
```

Al ejecutar este código, Python extraerá el contenido de `example.zip` en el directorio de trabajo actual. Como opción, podemos pasar un nombre de carpeta a `extractall()` para que extraiga los archivos a una carpeta distinta. Si la carpeta pasada a `extractall()` no existe, Python la creará. Por ejemplo, si sustituýésemos la llamada en 1 por `example_zip.extractall('C:\`

\spam), el código extraería los archivos de example.zip en una carpeta C:\spam recién creada.

El método `extract()` de los objetos `ZipFile` extraerá un solo archivo del ZIP. Continúa con el ejemplo del intérprete de comandos interactivo escribiendo esto:

```
> > > example_zip.extract('file1.txt')
```

```
'C:\\Users\\Al\\Desktop\\file1.txt'
```

```
> > > example_zip.extract('file1.txt', 'C:\\some\\new\\  
\\folders')
```

```
'C:\\some\\new\\folders\\file1.txt'
```

```
> > > example_zip.close()
```

La cadena que pasemos a `extract()` debe coincidir con una de las cadenas de la lista devuelta por `namelist()`. Como opción, podemos pasar un segundo argumento a `extract()` para extraer el archivo en una carpeta distinta del directorio de trabajo actual. Si este segundo argumento es una carpeta que no existe todavía, la creará Python.

Proyecto 5: Hacer una copia de seguridad de una carpeta en un archivo ZIP

Imagina que estás trabajando en un proyecto cuyos archivos guardas en una carpeta llamada `C:\Users\Al\AlsPythonBook`. Te preocupa perder el trabajo hecho, así que te gustaría crear “instantáneas” como archivos ZIP de la carpeta entera. También te conviene tener distintas versiones de estas instantáneas, así que necesitas que el nombre del archivo ZIP se incremente cada vez que se hace una nueva versión; por ejemplo, `AlsPythonBook_1.zip`, `AlsPythonBook_2.zip`, `AlsPythonBook_3.zip`, etc. Podrías hacerlo a mano, pero sería bastante tedioso y podrías equivocarte con la numeración de los nombres de los ZIP. Sería mucho más sencillo ejecutar un programa que haga esa aburrida tarea por ti. Para este proyecto, abre una ventana nueva en el editor de archivos y guarda como `backup_to_zip.py`.

Paso 1: averiguar el nombre del archivo ZIP

Vamos a poner el código de este programa en una función llamada `backup_to_zip()`. Así será más fácil copiar y pegar la función en otros programas de Python que necesiten esta funcionalidad. Al final del programa, llamaremos a la función para que haga la copia de seguridad. Empieza así tu programa:

```
# backup_to_zip.py – Copia una carpeta entera y su contenido en
# un archivo ZIP cuyo nombre de archivo se incrementa

import zipfile, os

from pathlib import Path

def backup_to_zip(folder):

    # Hace una copia de seguridad de todo el contenido de “folder” en
    un archivo ZIP.

    folder = Path(folder) # Se asegura de que folder es un objeto Path,
    no una cadena.

    # Averigua el nombre del archivo ZIP que debería usar este código,
    basándose en

    # los archivos que ya existen.

    1 number = 1

    2 while True:

        zip_filename = Path(folder.parts[-1] + '_' + str(number) + '.zip')

        if not zip_filename.exists():

            break

        number = number + 1

    3 # PENDIENTE: Crear el archivo ZIP.

    # PENDIENTE: Recorrer todo el árbol de la carpeta y comprimir los
    archivos de cada carpeta.
```

```
print('Done.')
```

```
backup_to_zip(Path.home() / 'spam')
```

Primero, importa los módulos zipfile y os. A continuación, define una función backup_to_zip() que tome solo un parámetro: folder. Este parámetro es una cadena u objeto Path a la carpeta cuyo contenido hay que copiar. La función determinará qué nombre usar para el archivo ZIP que creará. Después, creará el archivo, recorrerá la carpeta folder y añadirá cada subcarpeta y archivo al ZIP. Escribe comentarios PENDIENTE para estos pasos en el código fuente para acordarte de hacerlos más adelante 3. La primera tarea, nombrar el archivo ZIP, utiliza el nombre base de la ruta absoluta del archivo. Si la carpeta que se va a copiar es C:\Users\Al\spam, the ZIP, el nombre de archivo debería ser spam_N.zip, donde N será 1 la primera vez que ejecutes el programa, 2 la segunda y así sucesivamente.

Puedes determinar lo que debería ser N comprobando si spam_1.zip ya existe, luego si existe spam_2.zip, etc. Usa una variable number para N 1 y sigue incrementándola dentro del bucle que llama a exists() para comprobar si el archivo existe 2. El primer nombre de archivo no encontrado hará que se rompa el bucle porque ya tendrá el nombre para el nuevo ZIP.

Paso 2: crea el nuevo archivo ZIP

Ahora vamos a crear el archivo ZIP. Amplía tu programa de la siguiente manera:

```
# backup_to_zip.py - Copia una carpeta entera y su contenido en
```

```
# un archivo ZIP cuyo nombre de archivo se incrementa
```

```
--fragmento omitido--
```

```
# Crea el archivo ZIP.
```

```
print(f'Creating {zip_filename}...')
```

```
backup_zip = zipfile.ZipFile(zip_filename, 'w')
```


PENDIENTE: Recorrer todo el árbol de la carpeta y comprimir los archivos de cada carpeta.

```
print('Done.')
```

```
backup_to_zip(Path.home() / 'spam')
```

Ahora que el nombre del nuevo archivo ZIP está almacenado en la variable `zip_filename`, podemos llamar a `zipfile.ZipFile()` para crear realmente el archivo ZIP. Asegúrate de pasar 'w' como segundo argumento para abrir el archivo en modo escritura. También quitaremos el comentario PENDIENTE, ya que hemos terminado de escribir el código de esta sección.

Paso 3: recorrer el árbol de directorios

Ahora necesitamos la función `os.walk()` para la tarea de hacer una lista con todos los archivos de la carpeta y sus subcarpetas. Añade esto al programa:

```
# backup_to_zip.py - Copia una carpeta entera y su contenido en
```

```
# un archivo ZIP cuyo nombre de archivo se incrementa
```

```
-- fragmento omitido --
```

```
# Recorre todo el árbol de la carpeta y comprime los archivos de cada carpeta.
```

```
1 for folder_name, subfolders, filenames in os.walk(folder):
```

```
    folder_name = Path(folder_name)
```

```
    print(f'Adding files in folder {folder_name}...')
```

```
# Añade todos los archivos de esta carpeta al archivo ZIP.
```

```
2 for filename in filenames:
```

```
    print(f'Adding file {filename}...')
```

```
    backup_zip.write(folder_name / filename)
```

```
backup_zip.close()
```

```
print('Done.')
```

```
backup_to_zip(Path.home() / 'spam')
```

Utiliza `os.walk()` en un bucle `for` 1. En cada iteración, la función devolverá el nombre actual de la carpeta, las subcarpetas que hay en ella y todos los archivos. El bucle `for` anidado puede pasar por cada nombre de archivo de la lista 2. Todos ellos se añaden al archivo ZIP.

Cuando ejecutes este programa, debería producir una salida parecida a esta:

```
Creating spam_1.zip...
```

```
Adding files in spam...
```

```
Adding file file1.txt...
```

```
Done.
```

La segunda vez que lo ejecutes, pondrá todos los archivos de la carpeta `spam` en un ZIP llamado `spam_2.zip` y así sucesivamente.

Ideas para otros programas

Puedes recorrer un árbol de directorios y añadir archivos a ZIP comprimidos en otros muchos programas. Por ejemplo, podrías escribir programas que hagan lo siguiente:

- Recorrer un árbol de directorios y archivar solo los archivos que tengan una extensión determinada, como `.txt` o `.py`, y nada más.
- Recorrer un árbol de directorios y archivar todos los archivos salvo los `.txt` y los `.py`.
- Archivar solo las carpetas de un árbol de directorios que ocupen más espacio en el disco o hayan sido modificados desde el anterior archivo.

Resumen

Aunque seas un experimentado usuario de ordenadores, es probable que manejes los archivos manualmente con el ratón y el teclado. Los exploradores de archivos modernos hacen que sea fácil trabajar con unos cuantos archivos, pero a veces hay que hacer tareas que pueden llevarnos horas en el explorador,

Los módulos `os` y `shutil` ofrecen funciones para copiar, mover, renombrar y borrar archivos. Al borrar archivos, puede que te interese utilizar el módulo `send2trash` para mover archivos a la papelera de reciclaje en vez de eliminarlos definitivamente. Y, cuando escribas programas que manejen archivos, te conviene ejecutar un simulacro; anula con comentarios el código que copia, mueve, renombra o borra archivos y sustitúyelo por una llamada a `print()`. De ese modo, puedes ejecutar el programa y comprobar qué hará exactamente.

Con frecuencia, tendrás que hacer estas operaciones no solo con los archivos de una carpeta, sino con todas las subcarpetas de la carpeta y las subcarpetas de las subcarpetas, etc. La función `os.walk()` maneja este paseo por las carpetas por ti para que puedas concentrarte en lo que tiene que hacer tu programa con los archivos que hay en ellas.

El módulo `zipfile` proporciona una manera de comprimir y extraer ficheros en archivos `.zip` con Python. En combinación con las funciones de gestión de archivos de `os` y `shutil`, `zipfile` facilita el empaquetado de varios archivos de cualquier parte del disco duro. Estos archivos ZIP son mucho más fáciles de cargar en sitios web o enviar como adjuntos por correo electrónico que varios archivos separados.

Preguntas de práctica

1. ¿Cuál es la diferencia entre `shutil.copy()` y `shutil.copytree()`?
2. ¿Qué función sirve para renombrar archivos?
3. ¿Cuál es la diferencia entre las funciones de eliminación de los módulos `send2trash` y `shutil`?

4. Los objetos `ZipFile` tienen un método `close()`, igual que los objetos `File`. ¿Qué método de `ZipFile` equivale al método `open()` de los objetos `File`?

Programas de práctica

Para practicar, escribe programas que realicen las siguientes tareas.

Copia selectiva

Escribe un programa que recorra un árbol de directorios y busque archivos con una extensión determinada (como `.pdf` o `.jpg`). Copia estos archivos de su ubicación actual a una nueva carpeta.

Eliminar archivos innecesarios

No es raro que unos pocos archivos o carpetas innecesarios pero gigantes ocupen un montón de espacio en el disco duro. Si estás intentando liberar espacio, es más efectivo identificar primero los archivos más grandes que no hacen falta. Escribe un programa que recorra un árbol de directorios y busque archivos o carpetas excepcionalmente grandes, por ejemplo, los que tengan un tamaño superior a 100 MB. (Recuerda que, para obtener el tamaño de un archivo, puedes usar `os.path.getsize()` del módulo `os`). Imprime estos archivos con su ruta absoluta en la pantalla.

Renumerar archivos

Escribe un programa que busque todos los archivos con un prefijo determinado, como `spam001.txt`, `spam002.txt`, etc., en una única carpeta y localice cualquier laguna en la numeración (como si hubiese `spam001.txt` y `spam003.txt` pero no `spam002.txt`). Haz que el programa renombre todos los archivos posteriores para cerrar esa brecha.

Para crear estos archivos de ejemplo (que salten `spam042.txt`, `spam086.txt` y `spam103.txt`), ejecuta el siguiente código:

```
> > > for i in range(1, 121):
```

... if i not in (42, 86, 103):

... with open(f'spam{str(i).zfill(3)}.txt', 'w') as file:

... pass

...

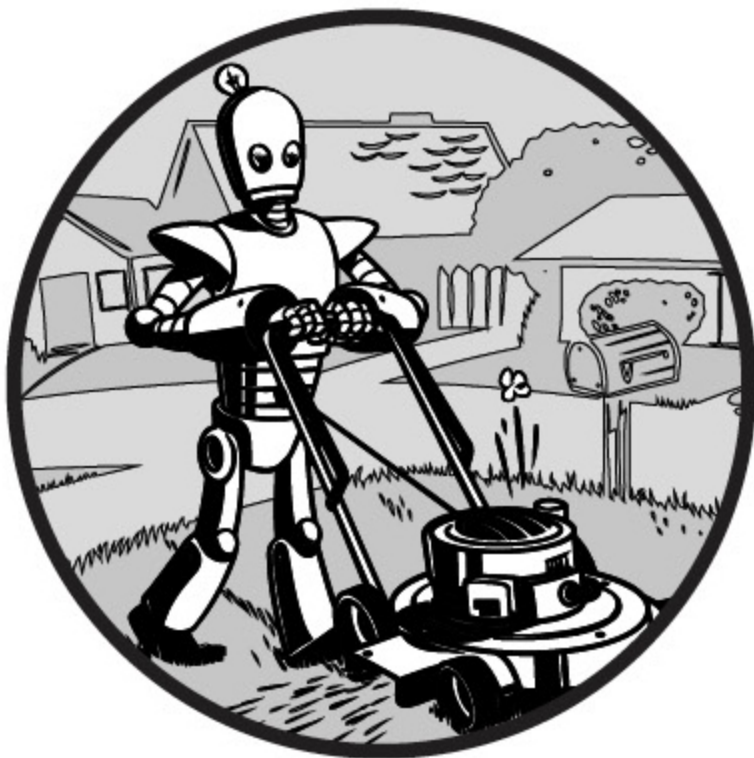
Como reto adicional, escribe otro programa que pueda insertar espacios en archivos numerados (y aumentar los números en los nombres de los archivos después del espacio) para que se pueda insertar un nuevo archivo.

Convertir fechas del formato americano al europeo

Imagina que tu jefe te envía por correo electrónico miles de archivos con fechas en formato americano (MM-DD-AAAA) en los nombres y necesita que los renombres al formato europeo (DD-MM-AAAA). Esta tediosa tarea podría llevarte todo el día si la hicieras a mano. En lugar de eso, escribe un programa que haga lo siguiente:

1. Buscar todos los nombres de archivo en el directorio de trabajo actual y en todas las subcarpetas que contengan fechas en formato americano. Utiliza la función `os.walk()` para recorrer las subcarpetas.
2. Utilizar expresiones regulares para identificar los nombres de archivo que contienen el patrón MM-DD-AAAA, por ejemplo, `spam12-31-1900.txt`. Supón que los meses y los días siempre se escriben con dos dígitos y que no existen archivos con nombres que no coincidan con fechas. (No encontrarás archivos con nombres como `99-99-9999.txt`).
3. Cuando se encuentre un nombre de archivo, renombrar el archivo intercambiando el mes y el día para que tenga el formato europeo. Utiliza la función `shutil.move()` para renombrar los archivos.

Diseñar y desplegar programas de línea de comandos



Hasta ahora, nos hemos centrado en ejecutar programas desde Mu (o el editor de código que uses). Este capítulo explica cómo ejecutar programas desde el terminal de línea de comandos. La línea de comandos puede intimidar un poco, con sus comandos crípticos y absoluta falta de una presentación amable con el usuario, así que la mayoría de usuarios la evitan. Pero tiene varias ventajas interesantes familiarizarse con ella y no es más difícil que cualquier programación con la que has practicado hasta el momento.

Una vez que hayas escrito un programa de Python para automatizar una tarea, tener que abrir Mu cada vez que quieras ejecutarlo puede

ser una lata. La línea de comandos es una forma más cómoda de ejecutar *scripts* de Python (sobre todo si compartes programas con amigos o compañeros que no tienen Mu, o incluso Python, instalado). En el desarrollo de software, el despliegue es el proceso de hacer que nuestro software pueda usarse fuera de nuestros editores de código.

Un programa con cualquier otro nombre

Este capítulo (y el mundo de la programación en general) usa distintos términos que significan programa o una ligera variación del concepto. Podemos llamar a cualquiera de los siguientes elementos “programa” y no estaríamos equivocados, pero hay algunas diferencias sutiles en los significados de estos nombres:

•**Programa:** Una pieza de software completa, grande o pequeña, con instrucciones que ejecuta un ordenador.

•**Script:** Un programa que un intérprete ejecuta desde su forma de código fuente en lugar de con la forma de código compilado. Es un término muy vago. Los programas de Python se denominan con frecuencia *scripts*, aunque el código Python puede compilarse como cualquier otro lenguaje (como veremos en “Compilar programas de Python con PyInstaller”).

•**Comando:** Un programa que se ejecuta a menudo desde un terminal basado en texto y no cuenta con una interfaz gráfica de usuario (GUI, *Graphical User Interface*). Toda la configuración se hace por adelantado, especificando los argumentos de la línea de comandos antes de ejecutar el comando (aunque los comandos interactivos pueden, en ocasiones, interrumpir su funcionamiento con preguntas de tipo “¿Estás seguro? S/N” para el usuario). Tanto `dir` como `ls`, explicados en “Los comandos `cd`, `pwd`, `dir` y `ls`”, son ejemplos de comandos.

•**Script de shell:** Un único archivo de texto que ejecuta cómodamente varios comandos de terminal agrupados en un solo lote. De esta forma, el usuario puede ejecutar un *script* de *shell* en lugar de introducir manualmente varios comandos de forma individual. En macOS y Linux, los archivos de *script* de *shell* tienen

la extensión `.sh` (o ninguna extensión), mientras que Windows utiliza el término “archivo por lotes” para los *scripts* de *shell* con la extensión `.bat`.

•**Aplicación:** Un programa que cuenta con una GUI y contiene muchas funciones relacionadas. Excel y Firefox son ejemplos de aplicaciones. Las aplicaciones suelen tener varios archivos que el programa instalador pone en nuestro ordenador (y que un programa desinstalador puede quitar), en lugar de consistir solo en un archivo ejecutable copiado en un ordenador.

•**App:** Un nombre habitual para las aplicaciones para teléfono móvil y tableta, pero también puede usarse para aplicaciones de escritorio.

•**Aplicación web:** Un programa que se ejecuta en un servidor web y con el que el usuario interactúa por Internet a través de un navegador.

Puedes ponerte tiquismiquis con las definiciones; estas explicaciones solo pretenden darte una idea general del uso de los términos. Si deseas familiarizarte con más terminología, mi libro *Beyond the Basic Stuff with Python* (No Starch Press, 2020) incluye un capítulo dedicado a la jerga de programación.

Utilizar el terminal

Hasta la década de los noventa, cuando Apple y Microsoft popularizaron los ordenadores con GUI que podían ejecutar varios programas a la vez, los programas se iniciaban desde una interfaz de línea de comandos (CLI, *Command Line Interface*) y solían estar limitados a entrada y salida basadas en texto. Es probable que hayas oído otros términos para hablar de las CLI, como intérprete de comandos, terminal, *shell* o consola. Los desarrolladores de software todavía utilizan las CLI y suelen tener varias ventanas de terminal abiertas en sus ordenadores al mismo tiempo. Aunque un terminal basado en texto no tenga los iconos, botones y gráficos de una GUI, es una manera efectiva de usar un ordenador cuando ya conoces varios comandos.

Haz lo siguiente para abrir una ventana de terminal:

- En Windows, haz clic en el botón Inicio (o pulsa la tecla **Windows**) y escribe **Símbolo del sistema** (o **PowerShell** o **Terminal** si los tienes instalados).

- En macOS, haz clic en el icono Spotlight de la esquina superior derecha (o pulsa **Comando-Barra espaciadora**) y escribe **Terminal**.

- En Ubuntu Linux, pulsa la tecla **Meta** para abrir el Dash y escribe **Terminal**. Como alternativa, puedes usar el atajo de teclado **Control-Alt-T**.

Igual que el intérprete de comandos interactivo tiene un *prompt* `> > >`, el terminal también mostrará un *prompt* para introducir comandos. En Windows, será la ruta completa de la carpeta en la que estás, seguida por un paréntesis angular (`>`):

`C:\Users\al> your commands go here`

En macOS, el *prompt* muestra tu nombre de usuario, el nombre de tu ordenador y el directorio de trabajo actual (con tu carpeta de inicio representada como `~` para abreviar), seguidos de un signo de porcentaje (%):

`al@Als-MacBook-Pro ~ % your commands go here`

En Ubuntu Linux, el *prompt* es similar al de macOS, salvo que este empieza con el nombre de usuario y una arroba (@):

`al@al-VirtualBox:$ your commands go here`

Aunque es más fácil ejecutar programas desde el menú Inicio (Windows) o Spotlight (macOS), también es posible iniciarlos desde el terminal. El intérprete de Python en sí es un programa que se ejecuta a menudo desde el terminal.

En este capítulo, asumiremos que el programa de Python que queremos ejecutar se llama `yourScript.py` y se encuentra en una carpeta `Scripts` dentro de nuestra carpeta de inicio. No hace falta abrir `Mu` para acceder al intérprete de comandos interactivo de Python. Desde una ventana de terminal, podemos introducir `python` en Windows o `python3` en macOS y Linux para iniciarlo. (Aparecerá

el conocido *prompt* > > >). Para ejecutar un archivo .py desde el terminal, introduce su ruta de archivos después de python o python3; puede ser la ruta absoluta, como python C:\Users\al\Scripts\yourScript.py, o la relativa: python yourScript.py, si el directorio de trabajo actual es C:\Users\al\Scripts, la misma carpeta en la que está yourScript.py.

Los comandos cd, pwd, dir y ls

Igual que todos los programas en ejecución tienen un directorio de trabajo actual (CWD, *Current Working Directory*) al que se asocian las rutas de archivo relativas, el terminal también cuenta con su directorio de trabajo actual. Puedes ver este CWD como parte del *prompt* o ejecutando pwd en macOS y Linux, o cd, sin argumento de línea de comandos, en Windows.

Los programas de Python pueden cambiar el CWD llamando a la función os.chdir(). En el terminal, podemos hacer lo mismo introduciendo el comando cd seguido de la ruta relativa o absoluta de la carpeta a la que queremos cambiar:

```
C:\Users\al> cd Desktop
```

```
C:\Users\al\Desktop> cd .
```

```
C:\Users\al> cd C:\Windows\System32
```

```
C:\Windows\System32>
```

En Windows, puede ser necesario un paso más para cambiar la letra de la unidad. No se puede cambiar la unidad con el comando cd. En su lugar, hay que introducir la letra de la unidad seguida de dos puntos y luego usar cd para cambiar de directorio en la unidad:

```
C:\Windows\System32> D:
```

```
D:\> cd backup
```

```
D:\backup>
```

El comando dir en Windows y ls en macOS y Linux hará una lista

con el contenido de la carpeta y las subcarpetas del CWD:

```
C:\Users\al> dir
```

--fragmento omitido--

```
08/26/2036 06:42 PM 171,304 _recursive-centaur.png
```

```
08/18/2035 11:25 AM 1,278 _viminfo
```

```
08/13/2035 12:58 AM <DIR> _pycache_
```

```
77 File(s) 83,805,114 bytes
```

```
108 Dir(s) 149,225,267,200 bytes free
```

Al navegar por el sistema de archivos del terminal, con frecuencia irás alternando entre `cd` para cambiar de directorio y `dir`/`ls` para ver el contenido del directorio. Puedes ver una lista con todos los archivos ejecutables del CWD ejecutando `dir *.exe` en Windows y `file * | grep executable` en macOS y Linux. Una vez dentro de la carpeta que contiene el programa, se puede ejecutar de la siguiente manera:

- En Windows, escribe el nombre del programa con o sin la extensión `.exe`: `example.exe`.
- En macOS y Linux, escribe `./` seguido del nombre del programa: `./example`.

Naturalmente, siempre podemos introducir la ruta absoluta de un programa: `C:\full\path\to\example.exe` o `/full/path/to/example`.

Si queremos abrir un archivo que no sea un programa, por ejemplo, un archivo de texto llamado `example.txt`, podemos hacerlo con la aplicación asociada introduciendo `example.txt` en Windows u `open example.txt` en macOS y Linux. Esto hace, desde el terminal, lo mismo que el doble clic en el icono de `example.txt` en una GUI. Si no hay aplicaciones establecidas para archivos `.txt`, el sistema operativo pedirá al usuario que seleccione una y la recordará para el futuro.

Todos los programas en ejecución, al margen del lenguaje en el que se hayan escrito, tienen un conjunto de variables de cadena llamado “variables de entorno”. Una de ellas es PATH, que contiene una lista de carpetas que el terminal comprueba al escribir el nombre de un programa. Por ejemplo, si introducimos python en Windows o python3 en macOS y Linux, el terminal busca un programa con ese nombre en las carpetas recogidas en PATH. Los sistemas operativos tienen reglas un poco diferentes para utilizar PATH:

- Windows comprueba primero el CWD en busca de un programa con ese nombre y, después, las carpetas de PATH.

- Linux y macOS buscan solo en las carpetas de PATH y no comprueban el CWD. Si quieres ejecutar un programa llamado example en el CWD, debes escribir ./example en lugar de example.

Para ver el contenido de la variable de entorno PATH, ejecuta echo %PATH% en Windows o echo \$PATH en macOS y Linux. El valor almacenado en PATH es una cadena larga compuesta por nombres de carpetas separados por punto y coma (en Windows) o dos puntos (en macOS y Linux). Por ejemplo, en Ubuntu Linux, la variable de entorno PATH puede tener este aspecto:

```
al@al-virtual-machine:~$ echo $PATH
```

```
/home/al/.local/bin:/home/al/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/
```

```
usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
```

Si introdujésemos python3 en el terminal de Linux con esta variable PATH, Linux buscaría un programa llamado python3 en la carpeta /home/al/.local/bin primero, después en /home/al/bin y así sucesivamente. Al final, encontraríamos python3 en /usr/bin y ejecutaríamos eso. La variable de entorno PATH es cómoda porque podemos simplemente soltar un programa ahí para evitarnos ir a la carpeta con cd cada vez que queramos ejecutarlo.

Observa que la ventana de terminal no busca subcarpetas en una

carpeta PATH. Si C:\Users\al\Scripts está en PATH, ejecutar spam.exe ejecutará un archivo C:\Users\ al\Scripts\spam.exe, pero no un archivo C:\Users\al\Scripts\eggs\spam.exe.

Editar PATH

Hasta ahora, es posible que hayas estado guardando tus archivos .py en la carpeta mu_code que utiliza el editor Mu por defecto. Sin embargo, te aconsejo crear una carpeta Scripts en tu carpeta de inicio. Si tu nombre de usuario fuese al, esta carpeta sería así:

- C:\Users\al\Scripts en Windows.
- /Users/al/Scripts en macOS.
- /home/al/Scripts en Ubuntu Linux.

Vamos a añadir esta carpeta a PATH.

Windows

Windows tiene dos conjuntos de variables de entorno: las del sistema (que se aplican a todos los usuarios) y las de usuario (que ignoran las variables de entorno del sistema y se aplican solo al usuario actual). Para editarlas, pulsa **Windows-R**, introduce **Sysdm.cpl**, haz clic en Ejecutar y, en el cuadro de diálogo Propiedades del sistema, selecciona la pestaña Opciones avanzadas y haz clic en Variables de entorno.

Selecciona Path en la lista de variables de usuario de la parte superior de la pantalla (no la lista de variables del sistema, que está debajo), haz clic en Editar, añade el nombre de la nueva carpeta **C:\Users\al\Scripts** en el campo de texto que aparece con un separador de punto y coma y haz clic en Aceptar.

macOS y Linux

Para añadir carpetas a las variables de entorno PATH, hay que editar el *script* de inicio del terminal. Este archivo se llama .zshrc en macOS y .bashrc en Linux. En ambos casos, se encuentra en la

carpeta de inicio y contiene comandos que se ejecutan siempre que se abre una nueva ventana de terminal. En macOS, añade lo siguiente al final del archivo .zshrc:

```
export PATH = /Users/al/Scripts:$PATH
```

En Linux, añade lo siguiente al final del archivo .bashrc:

```
export PATH = /home/al/Scripts:$PATH
```

Esta línea modifica PATH para todas las ventanas de terminal que abras a partir de ahora, así que el cambio no afectará a las ventanas que tengas abiertas ahora mismo.

Los comandos `which` y `where`

Si quieres averiguar en qué carpeta de la variable de entorno PATH se encuentra un programa, puedes ejecutar `which` en macOS y Linux y `where` en Windows. Por ejemplo, introduce el siguiente comando `which` en el terminal de macOS:

```
al@Als-MacBook-Pro ~ % which python3
```

```
/Library/Frameworks/Python.framework/Versions/3.13/bin/  
python3
```

En Windows, introduce el siguiente comando `where` en el terminal:

```
C:\Users\al > where python
```

```
C:\Users\al\AppData\Local\Programs\Python  
\Python313\python.exe
```

```
C:\Users\al\AppData\Local\Programs\Python  
\Python312\python.exe
```

El comando `where` muestra cada carpeta de PATH que contiene un programa llamado python. El de la carpeta superior es la versión que se ejecuta al introducir python. Los comandos `which` y `where` son útiles cuando no se está seguro de cómo está configurada PATH y se necesita encontrar la ubicación de un programa en concreto.

Entornos virtuales

Supongamos que tenemos dos programas de Python, uno que utiliza la versión 1.0 de un paquete y otro que emplea la 2.0 del mismo paquete. Python no puede tener instaladas dos versiones de un mismo paquete a la vez. Si la versión 2.0 no es compatible con la 1.0, habría que desinstalar una versión para instalar la otra cada vez que haya que cambiar de programa.

La solución de Python a este problema son los entornos virtuales: instalaciones separadas de Python que tienen su propio conjunto de paquetes de terceros instalados. En general, cada aplicación de Python que crees necesita su propio entorno virtual, pero puedes usar un solo entorno para todos los *scripts* pequeños que hagas mientras aprendes a programar. Python puede crear entornos virtuales con su módulo integrado `venv`. Para crear un entorno virtual, usa `cd` para seleccionar la carpeta `Scripts` y ejecuta `run python -m venv .venv` (con `python3` en macOS y Linux):

```
C:\Users\al>
```

```
C:\Users\al> cd Scripts
```

```
C:\Users\al\Scripts> python -m venv .venv
```

Esto crea los archivos del entorno virtual en una nueva carpeta llamada `.venv`. Puedes elegir el nombre de carpeta que quieras, pero `.venv` es convencional. Los archivos y carpetas cuyos nombres empiezan con un punto están ocultos, aunque puedes seguir las instrucciones de la introducción del libro para hacer que tu sistema operativo los muestre por defecto. Al ejecutar `python` o `python3` desde el terminal, podemos seguir ejecutando nuestro intérprete de instalación de Python original. Para utilizar la versión de Python del entorno virtual, hay que activarlo. Para hacerlo en Windows, ejecuta el *script* `C:\Users\al\Scripts\.venv\Scripts\activate.bat`:

```
C:\Users\al\Scripts> cd .venv\Scripts
```

```
C:\Users\al\Scripts\.venv\Scripts> activate.bat
```

```
(.venv) C:\Users\al\Scripts\.venv\Scripts> where python.exe
```

```
C:\Users\Al\Scripts\.venv\Scripts\python.exe
```

```
C:\Users\Al\AppData\Local\Programs\Python  
\Python313\python.exe
```

```
C:\Users\Al\AppData\Local\Programs\Python  
\Python312\python.exe
```

Ejecutar `where python.exe` después de activar el entorno virtual muestra que ejecutar `python` desde el terminal ejecutará el intérprete de Python en la carpeta `.venv\Scripts` y no en el Python del sistema (que veremos enseguida).

El equivalente a este *script* en macOS y Linux es `~/Scripts/.venv/bin/activate`, pero por los permisos de seguridad no podemos ejecutarlo directamente, sino que hay que ejecutar el comando `source activate`:

```
al@al-virtual-machine:~/Scripts$ cd .venv/bin
```

```
al@al-virtual-machine:~/Scripts/.venv/bin$ source activate
```

```
(.venv) al@al-virtual-machine:~/Scripts/.venv/bin$ which python3  
/home/al/Scripts/.venv/bin/python3
```

La activación cambia la variable de entorno `PATH` de manera que `python` o `python3` ejecute el intérprete de Python dentro de la carpeta `.venv` en lugar de la original. También cambia el *prompt* del terminal para incluir `(.venv)` de forma que sepamos que está activo el entorno virtual. Ejecutar `which python3` en el entorno virtual activado muestra que `python3` ejecuta el intérprete de Python en la carpeta `.venv/bin` recién creada. Estos cambios solo se aplican a la ventana de terminal actual; cualquier otra ventana existente o nueva no tendrá estos cambios en la variable de entorno ni el *prompt*. Esta instalación de Python tiene solo los paquetes predeterminados, no cuenta con ninguno de los paquetes que hayas podido instalar en la instalación de Python. Puedes confirmar esto ejecutando `python -m pip list` para ver una lista de los paquetes instalados:

```
(.venv) C:\Users\al\Scripts\.venv\Scripts> python -m pip list
```


Package

pp0

Scripts

La práctica estándar es crear un entorno virtual para cada proyecto de Python en el que se está trabajando, ya que cada proyecto podría tener sus propias dependencias de paquetes exclusivas. Sin embargo, en Windows, podemos ser un poco más laxos con los pequeños *scripts* aleatorios que escribimos en nuestra carpeta Scripts: pueden compartir el mismo entorno todos. Los sistemas operativos macOS y Linux tienen sus propios programas que dependen de la instalación de Python que viene con el sistema operativo. Con la instalación o actualización de paquetes para esta instalación de Python original, llamada Python del sistema, existe una pequeña posibilidad de que se introduzcan incompatibilidades que puedan provocar el fallo de estos programas. Ejecutar tus propios *scripts* con el Python del sistema está bien; instalar paquetes de terceros en el Python del sistema es un poco arriesgado, pero crear un entorno virtual en tu carpeta Scripts es una buena precaución para evitar instalar paquetes incompatibles.

Ten en cuenta que Mu dispone de su propio entorno virtual. Cuando pulsamos **F5** para ejecutar un *script* en Mu, no tendrá los paquetes instalados en el entorno virtual de la carpeta Scripts\venv. A medida que progrese con tus habilidades de programación, te resultará más fácil tener la ventana de Mu abierta para editar el código y una ventana de terminal abierta para ejecutarlo. Puedes cambiar la ventana activa rápidamente con el atajo **Alt-Tab** en Windows y Linux y **Comando-Tab** en macOS.

Para desactivar el entorno virtual, ejecuta `deactivate.bat` (en Windows) o `deactivate` (en macOS y Linux) en la misma carpeta que el *script* para activarlo. También puedes simplemente cerrar la ventana de terminal y abrir una nueva. Si quieres eliminar para siempre el entorno virtual con todos sus paquetes, borra la carpeta `.venv` con su contenido.

Los siguientes apartados explican cómo desplegar tu *script* una vez configurado el entorno virtual y añadida la carpeta Scripts a `PATH`.

Instalar paquetes de Python con pip

Python viene con un programa gestor de paquetes de línea de comandos llamado `pip` (escrito con minúscula salvo que esté empezando una frase). Aunque la librería estándar de Python viene con módulos como `sys`, `random` y `os`, hay cientos de miles de paquetes de terceros que podemos encontrar en PyPI, el Python Package Index, en <https://pypi.org>. En Python, un paquete es una colección de código Python disponible en PyPI, y un módulo es un archivo `.py` individual que contiene código Python. Los paquetes que instalamos de PyPI contienen módulos, que importamos con una declaración `import`.

Aunque `pip` en sí es un programa, es más fácil ejecutarlo a través del intérprete de Python ejecutando `python -m pip` en Windows o `python3 -m pip` en macOS y Linux, en vez de ejecutar el programa `pip` (en Windows) o `pip3` (en macOS y Linux) directamente. Así evitamos errores en los raros casos en los que haya varias instalaciones de Python, `PATH` esté mal configurada y `pip/pip3` se instale en un intérprete de Python distinto del que se ejecuta cuando introducimos `python/python3`.

No uses pip con Anaconda

Si tienes instalada la distribución Anaconda de Python en lugar de la habitual de <https://python.org>, deberías evitar utilizar `pip` en tus entornos `conda`. En su lugar, usa el gestor de paquete específico de `conda` a través del comando `conda`.

Para instalar un paquete desde PyPI, escribe lo siguiente en el terminal:

```
C:\Users\al> python -m pip install package_name
```

Recuerda ejecutar `python3` si trabajas con macOS o Linux en lugar de `python` para estos comandos. Ten en cuenta también que hay que ejecutar esto desde la ventana de terminal, no desde el intérprete interactivo de Python.

Para ver una lista de todos los paquetes que tienes instalados, junto con su número de versión, ejecuta `python -m pip list`:

```
C:\Users\al> python -m pip list
```

```
Package project location
```

```
-----
```

```
0117-2ph  
218312-cffi  
218312-cffi-bindings  
async-generator  
--fragmento omitido--  
wspoto
```

También puedes actualizar un paquete a su versión más reciente en PyPI ejecutando `python -m pip install -U nombre_del_paquete`, o instalar una versión en particular (por ejemplo, 1.17.4) ejecutando `python -m pip install nombre_del_paquete==1.17.4`.

Para desinstalar un paquete, ejecuta `python -m pip uninstall nombre_del_paquete`. Encontrarás más información sobre pip ejecutando `python -m pip --help`.

Instalar el paquete `automateboringstuff3`

Este libro habla de varios paquetes de terceros. Con el tiempo, los autores de estos paquetes pueden hacer actualizaciones que ya no sean compatibles con las versiones empleadas en el libro. Para garantizar que la información del libro es exacta, instala las versiones específicas enumeradas en el apéndice A. Instalar las últimas versiones funcionará bien a veces, pero hay que comprobar la documentación en línea del paquete para saber si hay cambios.

La forma más fácil de instalar todas las versiones correctas de los paquetes mencionados en este libro es instalar el paquete `automateboringstuff3` en un entorno virtual. Este paquete actúa como contenedor para todos los paquetes (y las versiones correspondientes) que aparecen en el libro. Desde un terminal de Windows, activa un entorno virtual y ejecuta:

```
python -m pip install automateboringstuff3
```

En macOS y Linux, activa un entorno virtual y ejecuta:

python3 -m pip install automateboringstuff3

Programas de Python autoconscientes

La librería estándar de Python no incluye ningún módulo que dé a los programas consciencia (todavía), pero hay varias variables integradas que pueden dar a un programa de Python información útil sobre sí mismo, el sistema operativo en el que está y el intérprete de Python que lo está ejecutando. El intérprete de Python configura estas variables automáticamente.

La variable `_file_` contiene la ruta del archivo `.py` como cadena. Por ejemplo, si yo ejecuto un archivo `yourScript.py` en mi carpeta de inicio, se evalúa a `'C:\Users\al\yourScript.py'`. Importar `from pathlib import Path` y llamar a `Path(_file_)` devuelve un objeto `Path` para este archivo. Esta información es útil si necesitamos localizar archivos en la carpeta del programa de Python. La variable `_file_` no existe cuando se ejecuta el intérprete interactivo de Python.

La variable `sys.executable` contiene la ruta completa y el archivo del propio programa intérprete de Python, y la variable `sys.version` contiene la cadena que aparece arriba en el intérprete de comandos interactivo con información sobre la versión del intérprete de Python.

Las variables `sys.version_info.major` y `sys.version_info.minor` contienen enteros de los números de versión mayor y menor del intérprete de Python. En mi portátil, que ejecuta la versión 3.13.1 de Python, son 3 y 13, respectivamente. También se puede pasar `sys.version_info` a la función `list()` para obtener información más específica: `list(sys.version_info)` devuelve `[3, 13, 1 'final', 0]` en mi portátil. Tener así información de la versión es mucho más fácil para trabajar que tener que extraerla de la cadena `sys.version`.

La variable `os.name` contiene la cadena `'nt'` si se ejecuta en Windows y `'posix'` si lo hace en macOS o Linux. Esto es útil si un *script* de Python necesita ejecutar distinto código dependiendo del sistema operativo en el que se esté ejecutando.

Para una identificación más específica del sistema operativo, la variable `sys.platform` contiene `'win32'` en Windows, `'darwin'` en

macOS y 'linux' en Linux.

Si necesitas información muy específica sobre la versión del sistema operativo y el tipo de CPU, el módulo integrado platform puede recuperar esta información. Este módulo tiene documentación en línea en <https://docs.python.org/3/library/platform.html>.

Si necesitas comprobar si un módulo está instalado, pon la declaración `import` en un bloque `try` y captura la excepción `ModuleNotFoundError`:

`try:`

`import nonexistentModule`

`except ModuleNotFoundError:`

`print("This code runs if nonexistentModule was not found.")`

Si el módulo es necesario para que el programa funcione, puedes poner un mensaje de error descriptivo aquí y llamar a `sys.exit()` para salir del programa. Eso será más útil para el usuario que un mensaje de error genérico.

Diseño de programas basados en texto

Antes de que lo habitual fuesen los sistemas operativos con GUI, todos los programas utilizaban texto para comunicarse con su usuario. Este libro se centra en crear pequeños programas útiles en vez de aplicaciones de software profesionales, por eso nuestros programas usan `print()` e `input()` a través de una interfaz de línea de comandos en lugar de las ventanas, botones y gráficos que ofrece una GUI.

Sin embargo, incluso cuando las aplicaciones de software se limitan a texto pueden ofrecer una interfaz similar a las GUI modernas. La figura 12.1 muestra Norton Commander, una aplicación para buscar en el sistema de archivos. Este tipo de aplicaciones se denominan TUI (*Text-based User Interface*, interfaz de usuario basada en texto).

D:\ Name	Size	Date	Time	D:\ Name	Name	Name
NC	PSUB-DIR4	2-12-23	1:59p	nc	fill	ncnet exe
				nc	hlp	ncnet msg
				nc	ico	ncpscr ip
				nc	msg	ncsf exe
				nc	plf	ncsf exe
				nc_exit	com	ncsl exe
				nc_exit	doc	ncsl msg
				ncclean	exe	ncview msg
				ncclean	ini	nczip exe
				ncdd	exe	nczip msg
				ncdd	msg	norton ini
				ncedit	exe	packer exe
				ncedit	msg	packer msg
				ncff	exe	packer set
				ncff	hlp	paraview exe
				ncff	msg	pc241b exe
				nclabel	exe	playwave exe
				ncmain	exe	q4view exe
NC	PSUB-DIR4	2-12-23	1:59p	uview.exe	165834	2-12-23 1:58p

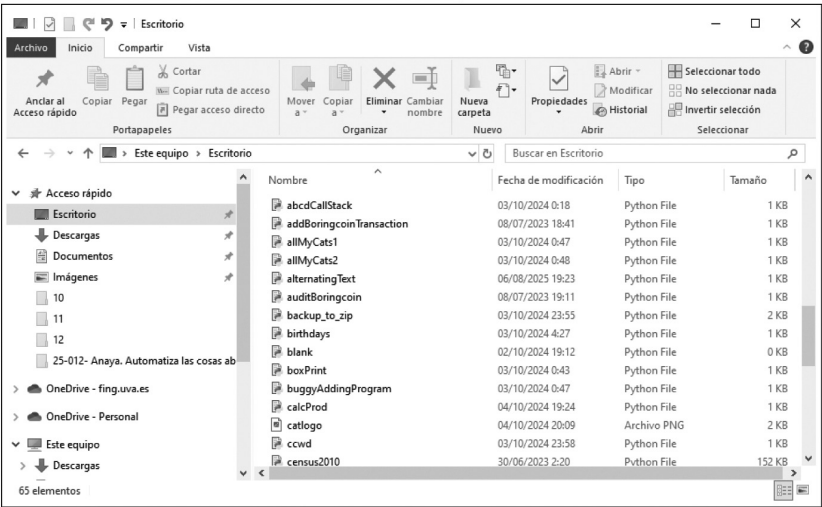


Figura 12.1. La interfaz basada en texto Norton Commander (arriba) y una aplicación con GUI moderna (abajo).

Incluso aunque no seas un desarrollador de software profesional, la ventaja de las interfaces de usuario basadas en texto es su sencillez. Este apartado describe varios enfoques de diseño que pueden adoptar tus programas para su interfaz de usuario.

Nombres de comandos cortos

Los usuarios suelen ejecutar programas basados en texto desde la línea de comandos en lugar de haciendo clic en un icono del Escritorio o del menú Inicio. A veces, estos comandos pueden parecer difíciles de entender. Cuando empecé a aprender a usar el sistema operativo Linux, me sorprendió descubrir que el comando copy de Windows que conocía bien se llamaba cp en Linux. El

nombre copy era mucho más legible que cp. ¿Merecía realmente la pena usar un nombre sucinto y críptico para ahorrarse escribir dos caracteres?

A medida que fui cogiendo experiencia con la línea de comandos, me di cuenta de que la respuesta era un sí rotundo. Leemos código fuente con más frecuencia que con la que lo escribimos, así que ayuda usar nombres verbosos para variables y funciones. Pero escribimos comandos en la línea de comandos con más frecuencia que con la que los leemos, así que también es cierto el enfoque contrario: los nombres de comandos cortos hacen que sea más fácil usar la línea de comandos y reducen la presión sobre nuestras muñecas.

Si tu programa es un comando que es probable que escribas decenas de veces al día, piensa un nombre corto para él. Puedes usar los comandos `which` y `where` para comprobar si ya tiene ese nombre otro programa. También puedes buscar en Internet si hay programas con ese nombre. Un nombre corto facilita mucho su uso.

Argumentos de línea de comandos

Para ejecutar un programa desde la línea de comandos, basta con introducir su nombre. Para los archivos de código fuente de Python `.py`, hay que ejecutar `python` (Windows) o `python3` (macOS y Linux) y después proporcionar el nombre del archivo `.py`, así: `python yourScript.py`.

El texto proporcionado después de un comando se llama argumento de línea de comandos. Estos argumentos se pasan a los comandos igual que los de las llamadas a funciones. Por ejemplo, el comando `ls` por sí solo muestra los archivos del CWD, pero, si ejecutamos `ls exampleFolder`, el argumento de línea de comandos `exampleFolder` dirá al comando `ls` que muestre los archivos de la carpeta `exampleFolder`. Los argumentos de línea de comandos permiten configurar el comportamiento del comando.

Los *scripts* de Python pueden acceder a los *scripts* de la línea de comandos dados al intérprete de Python desde la lista `sys.argv`. Por ejemplo, si introdujésemos `python3 yourScript.py hello world`, el

programa python3 recibiría los argumentos de la línea de comandos y los redirigiría al *script* de Python en la variable `sys.argv`. Esta variable contendría `['yourScript.py', 'hello', 'world']`. Observa que el primer elemento de `sys.argv` es el nombre de archivo del *script* de Python. Los demás argumentos se separan con espacios. Si necesitas incluir caracteres de espacio en un argumento de línea de comandos, ponlos entre comillas dobles al ejecutar el programa. Por ejemplo, `python3 yourScript.py "hello world"` establecería `sys.argv` en `['yourScript.py', 'hello world']`.

La principal utilidad de los argumentos de línea de comandos es que se puede especificar una amplia variedad de configuraciones antes de iniciar el programa. No hace falta pasar por un menú de configuración ni un proceso con varios pasos. Desafortunadamente, este enfoque implica que los argumentos de línea de comandos pueden llegar a ser increíblemente complicados e ilegibles. Si pasas `/?` después de cualquier comando de Windows o `--help` después de cualquier comando de macOS o Linux, encontrarás documentación sobre los argumentos de línea de comandos.

Si el conjunto de argumentos de línea de comandos que acepta tu programa es sencillo, será fácil hacer que el programa lea la lista `sys.argv` directamente. Sin embargo, a medida que añadas argumentos de línea de comandos, las combinaciones posibles se harán más difíciles de manejar. ¿Debería `python yourScript.py spam eggs` hacer lo mismo que `python yourScript.py eggs spam`? Si el usuario puede tener un argumento `cheese` o un argumento `bacon`, ¿qué pasa si pone los dos? Esta complejidad requiere escribir mucho código para gestionar los casos con varias aristas. En este punto, seguramente lo mejor para ti será usar el módulo integrado de Python `argparse` para manejar estas situaciones complicadas. Este módulo queda fuera del alcance de este libro, pero puedes leer la documentación en línea correspondiente en <https://docs.python.org/3/library/argparse.html>.

Entrada y salida del portapapeles

No hace falta depender de `input()` para leer textos de archivo o del teclado, también se puede usar el portapapeles para la entrada y la salida de texto de nuestros programas de Python. El módulo

multiplataforma pyperclip cuenta con una función `copy()` para mandar texto al portapapeles y una función `paste()` que devuelve el texto del portapapeles como una cadena. Este módulo es un paquete de terceros que se instala desde el terminal con `pip: python -m pip install pyperclip`. En Linux, también hay que ejecutar `sudo apt install xclip` para que funcione. Consulta el apéndice A para más instrucciones.

Todos tus programas con entrada/salida del portapapeles seguirán este diseño básico:

- 1.Importar el módulo `pyperclip`.
- 2.Llamar a `pyperclip.paste()` para obtener texto de entrada del portapapeles.
- 3.Realizar alguna tarea con el texto.
- 4.Copiar los resultados al portapapeles pasándoselos a `pyperclip.copy()`.

El proyecto “Añadir viñetas a marcado wiki” del capítulo 8 es un ejemplo de este tipo de programa. El diseño se vuelve especialmente útil una vez que lo hayas desplegado siguiendo las instrucciones de “Desplegar programas de Python”. Selecciona el texto de entrada, pulsa **Control-C** para copiarlo y ejecuta el programa. Los resultados estarán en el portapapeles, listos para pegarse donde haga falta. Más adelante en este capítulo, veremos dos proyectos (copiar el directorio de trabajo actual y la grabadora del portapapeles) que utilizan el portapapeles.

Texto en color con Bext

Podemos imprimir el texto en color usando el paquete de terceros `Bext`, integrado encima del paquete `Colorama` de Jonathan Hartley. Instala `Bext` con `pip` siguiendo las instrucciones del apéndice A. `Bext` solo funciona con programas ejecutados desde una ventana de terminal y no desde `Mu` u otros editores de código. Para que `print()` produzca texto en color, llama a las funciones `fg()` y `bg()` para cambiar el color del texto o del fondo con un argumento de cadena como `'black'`, `'red'`, `'green'`, `'yellow'`, `'blue'`, `'magenta'`, `'purple'`,

‘cyan’ o ‘white’. También puedes pasar ‘reset’ para cambiar el color de nuevo al predeterminado de la ventana de terminal. Para ver un ejemplo, escribe esto en el intérprete de comandos interactivo:

```
> > > import bext  
> > > bext.fg('red')  
> > > print("This text is red.")
```

This text is red.

```
> > > bext.bg('blue')  
> > > print("Red text on blue background is an ugly color  
scheme.")
```

Red text on blue background is an ugly color scheme.

```
> > > bext.fg('reset')  
> > > bext.bg('reset')  
> > > print("The text is normal again. Ah, much better.")
```

The text is normal again. Ah, much better.

Ten en cuenta que el usuario puede tener su ventana de terminal configurada en modo claro u oscuro, así que no se puede decir si el aspecto predeterminado del terminal es texto negro sobre fondo blanco o texto blanco sobre fondo negro. También deberías comedirtte en el uso del color: demasiado puede hacer que tu programa quede hortera o ilegible.

Bext también tiene algunas funciones de TUI limitadas, que incluyen las siguientes:

- bext.clear()**: Limpia la pantalla.
- bext.width()** y **bext.height()**: Devuelven la anchura (en columnas) y la altura (en filas) actual de la ventana de terminal.
- bext.hide()** y **bext.show()**: Ocultan o muestran el cursor.

- `bext.title(text)`: Cambia la barra de título de la ventana de terminal por la cadena de texto.
- `bext.goto(x, y)`: Mueve el cursor a la columna `x` y la fila `y` en el terminal, donde 0, 0 es la esquina superior izquierda.
- `bext.get_key()`: Espera a que el usuario pulse alguna tecla y devuelve una cadena describiendo la tecla.

Piensa en la función `bext.get_key()` como una versión de una sola tecla de `input()`. La cadena devuelta incluye 'a', 'z' y '5', pero también teclas como 'left', 'f1' y 'esc'. Las teclas **Tab** e **Intro** devuelven '\t' y '\n', respectivamente. Llama a `bext.get_key()` en el intérprete de comandos interactivo para probar las distintas teclas y ver sus valores de retorno.

Para una demostración de lo que puede hacer Bext, ejecuta el código fuente para el programa ASCII Art Fish Tank de <https://inventwithpython.com/projects/fishtank.py>. Primero, este programa usa `bext.clear()` para quitar todo el texto de la ventana de terminal. A continuación, llama a `bext.goto()` para colocar el cursor y a `bext.fg()` para cambiar el color antes de imprimir varios peces hechos con caracteres de texto, como `> < ))) * >`. Este programa está en mi libro *The Big Book of Small Python Projects* (No Starch Press, 2021).

Limpiar el terminal

La función `bext.clear()` es útil cuando queremos que el programa quite cualquier texto antes de ejecutarse. También podemos usarla para hacer una animación tipo filoscopio: llama a `clear()` para limpiar el terminal, usa llamadas a `print()` para rellenarlo con texto, haz una breve pausa con `time.sleep()` de Python y repite. Hay un *one-liner* de Python (programa de una línea de código para realizar una tarea específica) que borra la pantalla y puedes poner en tu propia función `clear()`:

```
import os

def clear():
```

```
os.system('cls' if os.name == 'nt' else 'clear')
```

Este código permite al programa limpiar la pantalla del terminal sin requerir la instalación del paquete Bext y solo funciona con *scripts* de Python ejecutados desde el terminal, no desde Mu ni ningún otro editor de código. La llamada a `os.system()`, sobre la que vimos más en el capítulo 11, ejecuta el programa `cls` (en Windows) o `clear` (en macOS y Linux). Esta extraña sintaxis es un ejemplo de expresiones condicionales de Python (llamadas operadores ternarios en otros lenguajes). La sintaxis es *valor1* *if condición* *else valor2*, lo que evalúa al valor 1 si la condición es `True` y al valor 2 si es `False`. En nuestro caso, la expresión condicional se evalúa a `'cls'` si la condición `os.name == 'nt'` es `True`; de lo contrario, se evalúa a `'clear'`. Las expresiones condicionales (y los *one-liners* en general) suelen producir código ilegible y es mejor evitarlas, pero este caso era bastante sencillo.

Notificaciones de sonido y texto

Los programas de terminal ya existían antes del rico sonido que ofrecen los ordenadores actuales. Hoy en día, no hay razón para que tus programas basados en texto sean mudos. Sin embargo, hay buenos motivos para mantener los sonidos al mínimo o excluirlos del todo. Los sonidos pueden dar notificaciones de que se ha completado una tarea o se ha producido un problema mientras el usuario está ocupado mirando otra ventana. Pero, al igual que con el texto de colores, es fácil pasarse con los sonidos hasta el punto de que resulten molestos. El usuario puede estar haciendo una tarea que ya implique reproducir sonidos, o podría estar en una reunión virtual que el sonido interrumpiría groseramente. Y, si el ordenador del usuario está silenciado, tampoco oíría la notificación.

Si solo necesitas reproducir un archivo de sonido sencillo, puedes usar el paquete de terceros `playsound3`. Una vez instalado, puedes reproducir un archivo de sonido llamando a la función `playsound()` del módulo `playsound3` y pasándole la ruta a un archivo MP3 o WAV. Descarga el archivo `hello.mp3` de <https://autbor.com/hello.mp3> (o usa un archivo tuyo) y escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > import playsound3
```

```
> > > playsound3.playsound('hello.mp3')
```

La función `playsound()` no volverá hasta que haya terminado de reproducirse el archivo de audio; es decir, la función se bloquea hasta que termine el sonido. Ten en cuenta que esto detiene el programa un rato si le das un archivo de sonido largo. Si `playsound()` lanza excepciones (lo que sucede si el nombre del archivo contiene caracteres raros, como un signo de igualdad), prueba a pasar un objeto `Path` del archivo en vez de una cadena.

Del mismo modo, puede que quieras limitar el texto producido por tu programa. Siguiendo la filosofía Unix de diseño de comandos, enviar la salida de texto de un comando a otro es más fácil si el comando solo muestra información relevante, ya que habría que filtrar el texto superfluo. Muchos comandos reducen al mínimo la salida de texto o no muestran nada y comunican el éxito o el error con el código de salida. (Hablaemos de los códigos de salida en el capítulo 19). Sin embargo, si no vas a mandar la salida a otro comando y, como usuario humano, prefieres ver más información, muchos comandos aceptan un argumento de línea de comandos `-v` o `--verbose` para habilitar este modo verboso. Otros comandos adoptan el enfoque contrario y llenan la salida con información, pero ofrecen un argumento de línea de comandos `-q` o `--quiet` para ofrecer un modo sin salida de texto. (Esto también puede servir para silenciar notificaciones sonoras). Lo mejor es silenciar el comportamiento determinado y hacer que `--verbose` o `--beep` habiliten el sonido o los pitidos.

Si tu programa no requiere este nivel de sofisticación, puedes ignorar esta consideración. Sin embargo, una vez que empieces a compartir tus programas con otros que pueden usarlos de maneras que no has previsto, ofrecer estas opciones ayuda mucho a que tus programas sean fáciles de usar.

Un programa corto: tormenta de nieve

Vamos a crear una animación de tormenta de nieve basada en texto. Nuestro programa utiliza caracteres de texto en bloque que ocupan

la mitad superior, la mitad inferior y todo el bloque de una celda de un solo carácter. Estos caracteres son devueltos como cadenas por chr(9600), chr(9604) y chr(9608), respectivamente, y nuestro programa los almacena en las constantes TOP, BOTTOM y FULL para que nuestro código sea más legible. Escribe el siguiente código en un archivo llamado snowstorm.py:

```
import os, random, time, sys

TOP = chr(9600) # El caracter 9600 es '█'
BOTTOM = chr(9604) # El caracter 9604 es '▣'
FULL = chr(9608) # El caracter 9608 es '▤'

# Establece la densidad de la tormenta de nieve en el argumento de
la línea de comandos:

DENSITY = 4 # La densidad predeterminada es 4%

if len(sys.argv) > 1:
    DENSITY = int(sys.argv[1])

def clear():
    os.system('cls' if os.name == 'nt' else 'clear')

while True:
    clear() # Limpia la ventana de terminal.

    # Pasa en bucle por cada fila y columna:
    for y in range(20):
        for x in range(40):
            if random.randint(0, 99) < DENSITY:
                # Imprime nieve:
```

```
print(random.choice([TOP, BOTTOM]), end = "")
```

else:

```
# Imprime espacio en blanco:
```

```
print(' ', end = "")
```

```
print() # Imprime un salto de línea.
```

```
# Imprime el suelo cubierto de nieve:
```

```
print(FULL * 40 + '\n' + FULL * 40)
```

```
print('(Ctrl-C to stop.)')
```

```
time.sleep(0.2) # Pausa un momento.
```

Primero, el programa importa los módulos `os`, `random`, `sys` y `time`. Estos módulos están en la librería estándar de Python y no requieren instalar paquetes de terceros. Después, el programa configura las constantes `TOP`, `BOTTOM` y `FULL` con valores de retorno de `chr()`. El programa usa estos nombres de constantes porque son más fáciles de entender que los números 9600, 9604 y 9608.

El usuario puede especificar la densidad de la tormenta de nieve proporcionando un argumento de línea de comandos. Si no se da ninguno, `sys.argv` se establece en `['snowstorm.py']` y el programa deja `DENSITY` en 4. Pero, si el usuario ejecutase el programa con, por ejemplo, `python snowstorm.py 20`, `sys.argv` se establecería en `['snowstorm.py', '20']` y el programa actualizaría `DENSITY` como `int(sys.argv[1])`, o 20. Entonces el usuario podría modificar el comportamiento de la tormenta de nieve sin tener que cambiar el código fuente.

Dentro de un bucle `while` infinito, este programa primero limpia la pantalla con el *one-liner* `cls/clear` y, a continuación, usa dos bucles `for` anidados para pasar por cada fila y columna de un espacio de 40×20 en el terminal. (Puedes aumentar o reducir estos números para cambiar el tamaño de tu tormenta de nieve). En cada fila y columna, el programa imprime un solo carácter: un carácter `TOP` o

BOTTOM seleccionado aleatoriamente representa nieve o un espacio en blanco. (Solo el cuatro por ciento de los caracteres no son espacios en blanco por defecto). Estas llamadas a `print()` pasan el argumento `end = "` para que `print()` no imprima automáticamente un salto de línea después de cada llamada. El programa imprime este salto de línea llamando a `print()` sin argumentos al acabar una fila. Después de los bucles for anidados, el programa imprime dos filas de 40 caracteres FULL para representar el suelo, junto con un recordatorio de que el usuario puede pulsar **Control-C** para detener el programa. Todo este código produce un “fotograma” de la animación de la tormenta de nieve y `time.sleep(0.2)` lo mantiene momentáneamente antes de que la ejecución vuelva al principio del bucle para borrar el terminal y reiniciar el proceso una y otra vez.

He elegido la animación de la tormenta de nieve porque es divertida, más que por que sea práctica, es como hacer una bola de nieve con el terminal. Una aplicación más práctica de esta técnica es crear una *app* de cuadro de mando: un programa que se ejecute en una ventana de terminal que dejes abierta para tener información de un vistazo. Este programa imprime información relevante y después borra y vuelve a imprimir la información actualizada cada segundo, minuto, hora o en cualquier otro intervalo.

Mensajes emergentes con PyMsgBox

Aunque diseñar una GUI completa para tu programa requiere aprender una librería de código entera, Tkinter, wxPython o PyQt, puedes añadir pequeños cuadros de diálogo con GUI a tu programa con el paquete PyMsgBox. Se trata de un paquete de terceros que puedes instalar ejecutando `pip install pymsgbox` desde el terminal. PyMsgBox permite crear diálogos con Tkinter, que viene con Python en Windows y macOS. En Ubuntu Linux, tendrás que instalar primero Tkinter ejecutando `sudo apt install python3-tk` en el terminal. Encontrarás las instrucciones en el apéndice A.

PyMsgBox tiene funciones con nombres que recuerdan a las funciones de cuadro de diálogo de JavaScript:

- `pymsgbox.alert(text)`: Muestra un mensaje hasta que el usuario

hace clic en Aceptar, después devuelve la cadena ‘OK’.

- `pymsgbox.confirm(text)`: Muestra un mensaje hasta que el usuario hace clic en Aceptar o Cancelar, después devuelve ‘OK’ o ‘Cancel’.

- `pymsgbox.prompt(text)`: Muestra un mensaje con un campo de texto y devuelve el texto que escriba el usuario o None si ha hecho clic en Cancelar.

- `pymsgbox.password(text)`: Es lo mismo que `pymsgbox.prompt()`, pero el texto del usuario se marca con asteriscos.

Estas funciones no volverán hasta que el usuario haga clic en Aceptar, Cancelar o X (cerrar). Si tu programa solo requiere una notificación o entrada de usuario ocasional, los cuadros de diálogo de PyMsgBox pueden ser un buen sustituto de `print()` e `input()`.

Desplegar programas de Python

Cuando tu programa de Python esté terminado, puede que no quieras ejecutar Mu, cargar el archivo .py y hacer clic en Ejecutar cada vez que quieras ejecutarlo. Este apartado explica cómo desplegar tu programa de Python para poder ejecutarlo con el menor número de pulsaciones de teclas posible.

Asegúrate de seguir los pasos de “La variable de entorno PATH” para añadir una carpeta Scripts a tu variable de entorno PATH. Como mi nombre de usuario es al, mi ruta a esta carpeta es C: \Users\al\Scripts en Windows, /Users/al/Scripts en macOS y /home/al/Scripts en Linux. Además, tendrás que configurar un entorno virtual para tus *scripts* de Python como se describe a continuación.

Windows

En Windows, puedes abrir el cuadro de diálogo Ejecutar pulsando **Windows-R** al mismo tiempo (o haciendo clic con el botón secundario en el botón del menú Inicio y seleccionando Ejecutar). Se abrirá una pequeña ventana que actúa como terminal de un solo uso: en ella, puedes ejecutar un solo comando. Para ejecutar tus

scripts de Python desde aquí, necesitas hacer esto:

1. Poner el *script* de Python `yourScript.py` en tu carpeta `Scripts`.
2. Crear un archivo por lotes `yourScript.bat` en la carpeta `Scripts` para ejecutar el *script* de Python.

Los archivos por lotes contienen comandos de terminal que se pueden ejecutar juntos ejecutando el archivo por lotes. Tienen una extensión de archivo `.bat` y son similares a los *scripts* para consola de Linux o los *scripts* `.command` de macOS. Si pones un archivo por lotes llamado `yourScript.bat` en una carpeta `PATH`, puedes ejecutarlo desde el cuadro de diálogo Ejecutar introduciendo `yourScript`. En Windows, no hace falta introducir la extensión para ejecutar archivos `.bat` o `.exe`.

El contenido de un archivo por lotes es texto simple, como el de un archivo `.py`, así que se puede crear archivos por lotes con Mu o un editor de texto como el Bloc de notas. Los archivos por lotes contienen un comando por línea. Para ejecutar un archivo `yourScript.py` ubicado en una carpeta `C:\Users\al\Scripts`, crea un archivo llamado `yourScript.bat` con el siguiente contenido:

```
@call %HOMEDRIVE%%HOMEPATH%\Scripts\.venv\Scripts
\activate.bat
```

```
@python %HOMEDRIVE%%HOMEPATH%\Scripts\yourScript.py
%*
```

```
@pause
```

```
@deactivate
```

Los archivos por lotes pueden tener cualquier nombre, pero es más fácil recordarlo si se llama igual que el *script* de Python. Este archivo por lotes ejecuta tres comandos. El primero activa el entorno virtual creado para la carpeta `Scripts`. El símbolo `@` al principio hace que el comando en sí no aparezca en la ventana de terminal. La variable de entorno `%HOMEDRIVE%` es `'C:'` y la variable de entorno `%HOMEPATH%` es la ruta a tu carpeta de inicio, como `'\Users\al'`. ([~ no representa la carpeta de inicio en

Windows). Cuando se combinan, producen la ruta al *script* de activación del entorno virtual, independientemente del nombre de usuario. (Esto es útil si compartes estos archivos con un compañero para que los ejecute en su equipo). Observa que la llamada es necesaria; si un archivo por lotes (como yourScript.bat) ejecuta otro archivo por lotes (como activate.bat) sin llamada, el resto de comandos del primer archivo por lotes no se ejecutarán.

A continuación, el archivo por lotes ejecuta python.exe, que a su vez ejecuta yourScript.py. %* hace que el archivo por lotes reenvíe cualquier argumento de línea de comandos que reciba a tu programa de Python. Es buena idea incluir siempre el %* por si más adelante añades argumentos de línea de comandos a tu programa de Python.

El tercer comando ejecuta el comando pause, que hace que Windows muestre Pulse cualquier tecla para continuar y espera a que el usuario pulse alguna tecla. Esto impide que la ventana de terminal se cierre de inmediato en cuanto termina el programa de Python, de manera que puedas ver la salida impresa que quede. Si el programa no tiene salida impresa, puedes omitir esta línea. Por último, la línea @deactivate desactiva el entorno virtual por si has ejecutado el archivo por lotes desde el terminal y esa ventana se queda abierta después de terminar el programa de Python.

Con el archivo por lotes ya configurado, puedes ejecutar tu *script* de Python pulsando **Windows-R** para abrir el cuadro de diálogo Ejecutar e introducir yourScript (seguido de cualquier argumento de línea de comandos) para ejecutar el *script* yourScript.bat. O, si tienes abierta una ventana de terminal, puedes escribir yourScript en el terminal desde cualquier carpeta. Esto es bastante más rápido que ejecutarlo desde un editor de código como Mu.

Si creas otros *scripts* de Python, puedes reutilizar este archivo por lotes. Basta con copiar el archivo con un nombre nuevo y cambiar el nombre de archivo yourScript.py por el del nuevo *script*. El resto se queda igual.

Pulsar **Comando-Barra espaciadora** simultáneamente en macOS abre el Spotlight, permitiéndonos introducir el nombre de un programa para ejecutar. Para añadir tus propios *scripts* de Python a Spotlight, debes hacer lo siguiente:

1. Poner el *script* de Python `yourScript.py` en tu carpeta `Scripts`.
2. Crear un archivo de texto `yourScript.command` para ejecutar el *script* de Python.
3. Ejecutar `chmod u + x yourScript .command` para añadir permisos de ejecución al archivo `yourScript.command`.

Una vez que tengas el *script* `.py` de Python en tu carpeta `Scripts`, como `/Users/al/Scripts`, crea un archivo de texto llamado `yourScript.command` en la carpeta `Scripts` con el siguiente contenido:

```
source ~/Scripts/.venv/bin/activate
```

```
python3 ~/Scripts/yourScript.py
```

```
deactivate
```

`~` representa la carpeta de inicio, como `/Users/al`. La primera línea activa el entorno virtual y la segunda ejecuta el *script* de Python utilizando la instalación de Python del entorno virtual.

Por último, en un terminal, usa `cd` para cambiar a `~/Scripts` y ejecuta el comando `chmod u + x yourScript.command`. Esto añade permisos de ejecución para que puedas ejecutar el *script* desde Spotlight. Ahora podrás ejecutar rápidamente el *script* de Python pulsando **Comando-Barra espaciadora** e introduciendo `yourScript.command`. (Spotlight debería autocompletar el nombre del archivo cuando escribes los primeros caracteres). También podrás ejecutar tu *script* de Python desde el terminal introduciendo `yourScript.command`.

El archivo `yourScript.command` hace falta porque, si intentas ejecutar `yourScript.py` desde Spotlight, este ve la extensión `.py` y asume que quieres abrir el archivo en Mu u otro editor de código en vez de ejecutarlo directamente.

Si usas TextEdit de macOS para crear el archivo `yourScript.command`, asegúrate de guardarlo como texto simple pulsando **Mayús-Comando-T** (o haciendo clic en Formato > Hacer texto sin formato). TextEdit también intentará ayudarte poniendo `python3` como `Python3`, con mayúscula, lo que produce un error en Spotlight.

Por desgracia, Spotlight no tiene forma de dejar al usuario pasar argumentos de línea de comandos al *script* de Python. Cualquier argumento de línea de comandos debe escribirse preventivamente en el archivo `.command`.

Ubuntu Linux

El Dash Ubuntu Linux puede abrirse pulsando la tecla **Meta** e introduciendo el nombre del programa que se quiere ejecutar. Para añadir tu *script* de Python a Dash, debes hacer lo siguiente:

1. Pon el *script* de Python `yourScript.py` en la carpeta `Scripts`.
2. Crea un *script* de *shell* llamado `yourScript` para activar el entorno virtual y ejecutar tu *script* de Python.
3. Ejecuta el comando `chmod u+x yourScript` para añadir permisos de ejecución al *script* de *shell*.
4. Crea un archivo `yourScript.desktop` en la carpeta `~/.local/share/applications` para ejecutar el *script* de *shell* desde Dash.

Una vez que tengas tu *script* `.py` de Python en tu carpeta `Scripts`, como `/home/al/Scripts`, crea un archivo de texto llamado `yourScript` (sin extensión de archivo) en la carpeta `Scripts` con el siguiente contenido:

```
#!/usr/bin/env bash

source ~/Scripts/.venv/bin/activate

python3 ~/Scripts/yourScript.py

read -p "Press any key to continue..." -n1 -s
```

deactivate

~ representa la carpeta de inicio, como /Users/al. La primera línea identifica un *script* de *shell*. No hace falta la extensión .sh para este archivo.

La segunda línea activa el entorno virtual y la tercera ejecuta el *script* de Python usando la instalación de Python en el entorno virtual. El comando read hace que el terminal muestre Press any key to continue y espere a que el usuario pulse alguna tecla. Esto evita que la ventana de terminal se cierre de inmediato cuando el programa de Python termine para que puedas ver la salida impresa restante. Si el programa no tiene salida impresa, puedes omitir esta línea.

Después de crear este *script* de *shell*, cambia con cd a ~/Scripts y ejecuta el comando chmod u+x yourScript. Esto le añade permisos de ejecución para que puedas ejecutarlo. En este punto, también podrás ejecutar tu *script* de Python desde el terminal introduciendo yourScript. Para ejecutarlo desde Dash, debes crear un archivo yourScript.desktop.

En Mu u otro editor de texto, como gedit, crea un archivo yourScript.desktop con el siguiente contenido:

[Desktop Entry]

Name = yourScript

Exec = gnome-terminal -- /home/al/Scripts/yourScript

Type = Application

Guarda este archivo en la carpeta /home/al/.local/share/applications (sustituyendo al por tu nombre de usuario) como yourScript.desktop. (Observa que el campo Exec requiere escribir /home/al, no sirve poner el valor con ~ en este archivo). Si tu editor de texto no muestra la carpeta .local (porque las carpetas que empiezan con un punto se consideran ocultas), pulsa **Control-H** en el cuadro de diálogo Guardar archivo para mostrar los elementos ocultos.

Ahora podrás ejecutar rápidamente el *script* de Python pulsando **Meta** para abrir Dash e introduciendo yourScript. Dash debería autocompletar el nombre por ti. El texto yourScript del campo Nombre de yourScript.desktop aparecerá en Dash y puede ser cualquier cosa, pero conviene ponerle el mismo nombre de yourScript.py.

Ahora vamos a crear dos programas siguiendo los principios de este capítulo y a desplegarlos para un uso fácil.

Un programa corto: copiar el directorio de trabajo actual

Mientras que el comando `pwd` en macOS y Linux imprime el directorio de trabajo actual, a veces es útil copiar este valor en el portapapeles para pegarlo en otro sitio. Por ejemplo, en Windows, siempre estoy copiando el directorio de trabajo actual desde el terminal para poder pegarlo en el cuadro de diálogo Guardar como para guardar un archivo en el mismo directorio. Aunque podría usar el ratón para seleccionar el directorio del *prompt* de Windows para copiarlo (o, en macOS y Linux, ejecutar el comando `pwd` para imprimir el directorio de trabajo y seleccionar ese texto para copiarlo), eso requiere varios pasos para lo que podría ser un proceso de un solo paso.

La idea es escribir un programa llamado `ccwd`. Primero, introduciré `where ccwd` en Windows y `which ccwd` en macOS y Linux para asegurarme de que no hay ya un comando con el mismo nombre y luego tal vez haga una búsqueda rápida en Internet de **ccwd** para asegurarme. El nombre es corto de escribir y único.

Como característica adicional, supongamos que el directorio de trabajo actual del terminal era `C:\Users\al\Scripts`, pero lo que yo quería copiar al portapapeles era `C:\Users\al`. Podría ejecutar el comando `cd ..`, luego `ccwd` y después `cd Scripts` para volver a `C:\Users\al\Scripts`. Pero sería más fácil pasar una ruta relativa a `ccwd` como argumento de línea de comandos. Por ejemplo, `ccwd ..` copiaría `C:\Users\al` al portapapeles cuando el directorio de trabajo actual fuese `C:\Users\al\Scripts`. No hace falta especificar este argumento de línea de comandos (el programa va por defecto al directorio de trabajo actual si no se da ninguno), pero está

disponible como función si el usuario lo necesita. Estas pequeñas mejoras pueden parecer triviales, pero las tiendas en línea tienen funciones de compra con un clic en su sitio web porque saben que cualquier paso hacia la comodidad puede tener un efecto importante.

Utilizaremos el paquete Pyperclip para gestionar el portapapeles, así que asegúrate de instalarlo en el entorno virtual de la carpeta Scripts. Crea un nuevo archivo en Mu y escribe el siguiente contenido:

```
import pyperclip, os, sys

if len(sys.argv) > 1:

    os.chdir(sys.argv[1])

pyperclip.copy(os.getcwd())
```

Guarda este programa como ccwd.py en la carpeta Scripts de tu carpeta de inicio.

La primera línea importa los módulos que necesita el programa. La segunda comprueba si se pasan al programa argumentos de línea de comandos. Recuerda que sys.argv es una lista que siempre contiene al menos una cadena: el nombre 'ccwd.py' del *script*. Si tiene más de una cadena, sabemos que el usuario suministró argumentos de línea de comandos al programa. En ese caso, la tercera línea cambia el directorio de trabajo actual. Ten en cuenta que cada programa tiene su propia configuración de directorio de trabajo actual y cambiarla con os.chdir() no cambia el directorio del terminal que ejecuta el programa. Por último, la cuarta línea copia el directorio de trabajo actual en el portapapeles.

El programa está terminado, pero, para ejecutarlo desde un terminal, tenemos que introducir su ruta completa, algo como python C:\Users\al\Scripts\ccwd.py. Eso es mucho escribir y no cumple el propósito de tener un *script* rápido y fácil para copiar el directorio de trabajo actual al portapapeles. Para mejorar el proceso, vamos a ver los pasos de despliegue para este programa en cada sistema operativo.

Windows

Guarda el archivo de Python como `C:\Users\al\Scripts\ccwd.py` (cambiando al por tu nombre de usuario). En la misma carpeta `Scripts`, crea un archivo `ccwd.bat` con el siguiente contenido:

```
@call %HOMEDRIVE%%HOMEPATH%\Scripts\.venv\Scripts
\activate.bat
```

```
@python %HOMEDRIVE%%HOMEPATH%\Scripts\ccwd.py %*
```

```
@deactivate
```

Este archivo por lotes no tiene la línea `@pause` porque no tiene salida de `print()`. Ya puedes ejecutar el programa desde el terminal en cualquier carpeta ejecutando `ccwd`:

```
C:\Users\al> ccwd
```

```
C:\Users\al>
```

Ahora mismo, `'C:\Users\al'` está en el portapapeles.

macOS

Guarda el archivo de Python como `/Users/al/Scripts/ccwd.py` (cambiando al por tu nombre de usuario). En la misma carpeta `Scripts`, crea un archivo de texto con el nombre `ccwd.command` y el siguiente contenido:

```
source ~/Scripts/.venv/bin/activate
```

```
python3 ~/Scripts/ccwd.py
```

```
deactivate
```

A continuación, en un terminal, usa `cd` para pasar a la carpeta `Scripts` y ejecuta `chmod u+x ccwd.command`:

```
al@Als-MacBook-Pro ~ % cd ~/Scripts
```

```
al@Als-MacBook-Pro Scripts % chmod u+x ccwd.command
```

Ya puedes ejecutar este programa desde el terminal en cualquier carpeta con `ccwd.command`:

```
al@Als-MacBook-Pro ~ % ccwd.command
```

```
al@Als-MacBook-Pro ~ %
```

‘/Users/al’ ya debería estar en el portapapeles.

Ubuntu Linux

Guarda el archivo de Python como `/home/al/Scripts/ccwd.py` (cambiando al por tu nombre de usuario). En la misma carpeta `Scripts`, crea un archivo de texto llamado `ccwd` con el siguiente contenido:

```
#!/usr/bin/env bash
```

```
source ~/Scripts/.venv/bin/activate
```

```
python3 ~/Scripts/ccwd.py
```

```
deactivate
```

Este *script* de *shell* `ccwd` no requiere la línea `read -p "Press any key to continue..." -n1 -s`, porque se ejecutará solo desde el terminal y no desde Dash, y la ventana de terminal no desaparecerá tras ejecutar el *script* de Python.

A continuación, en un terminal, usa `cd` para pasar a la carpeta `Scripts` y ejecuta `chmod u+x ccwd`:

```
al@al-VirtualBox:~$ cd ~/Scripts
```

```
al@al-VirtualBox:~/Scripts$ chmod u+x ccwd
```

Ya puedes ejecutar el programa desde el terminal en cualquier carpeta ejecutando `ccwd`:

```
al@al-VirtualBox:~$ ccwd
```

```
al@al-VirtualBox:~$
```

En este punto, ‘/home/al’ debería estar en el portapapeles.

Un programa corto: grabadora del portapapeles

Supongamos que parte de tu trabajo consiste en copiar URL de una página web y pegarlas en una hoja de cálculo. (En el próximo capítulo veremos cómo extraer todos los enlaces del código fuente HTML de la página, pero imagina que solo tienes que copiar algunas y un humano debe decidir cuáles, caso por caso). Podrías seguir estos pasos:

- 1.Hacer clic con el botón secundario en un enlace en un navegador web.
- 2.Seleccionar Copiar enlace en el menú contextual.
- 3.Cambiar a la aplicación de la hoja de cálculo.
- 4.Pulsar **Control-V** para pegar el enlace.
- 5.Volver al navegador web.

Es una tarea aburrida, sobre todo si la página tiene decenas o cientos de enlaces. Vamos a crear un pequeño programa de grabación en el portapapeles para hacerlo más rápido. Desplegaremos el programa en nuestro ordenador para poder ejecutarlo cómodamente cuando lo necesitemos. Nuestro programa monitorizará el portapapeles para ver si se ha copiado texto nuevo y, si es el caso, lo imprimirá en la pantalla del terminal. De ese modo, podemos convertir nuestro proceso de cinco pasos en uno de dos:

- 1.Hacer clic con el botón secundario en un enlace en un navegador web.
- 2.Seleccionar Copiar enlace en el menú contextual.

Después, el usuario puede simplemente copiar el texto de la grabadora del portapapeles de la ventana de terminal y pegarlo en la hoja de cálculo de una sola vez. Escribe lo siguiente en el editor de archivos y guarda como cliprec.py:

```

import pyperclip, time

print('Recording clipboard... (Ctrl-C to stop)')

previous_content = ''

try:

    while True:

        content = pyperclip.paste() # Coge el contenido del portapapeles.

        if content != previous_content:

            # Si es distinto del anterior, lo imprime:

            print(content)

            previous_content = content

            time.sleep(0.01) # Pausa para evitar el acaparamiento de CPU.

except KeyboardInterrupt:

    pass

```

Echemos un vistazo a cada parte del programa, empezando desde el principio:

```

import pyperclip, time

print('Recording clipboard... (Ctrl-C to stop)')

previous_content = ''

```

Este programa copia y pega texto desde el portapapeles, así que necesitamos importar el módulo `pyperclip`. También importaremos el módulo `time` para usar su función `sleep()`. El programa muestra un mensaje rápido para decir que se está ejecutando y recuerda al usuario que **Control-C** hace que el programa se detenga. El programa sabrá que el contenido del portapapeles ha cambiado llevando un registro de lo que había antes con una variable `previous_content`, que está inicialmente configurada como una

cadena en blanco.

try:

while True:

content = pyperclip.paste() # Coge el contenido del portapapeles.

El grueso del programa está en un bucle while infinito, que se encuentra dentro de un bloque try. Cuando el usuario pulsa **Control-C**, Python lanza una excepción KeyboardInterrupt, haciendo que la ejecución se mueva al bloque except al final del código fuente.

Este bucle monitoriza continuamente el contenido del portapapeles y anota cada vez que el usuario copia texto nuevo en él. Dentro del bucle, el primer paso es recoger el texto del portapapeles llamando a pyperclip.paste():

if content != previous_content:

Si es distinto del anterior, lo imprime:

print(content)

previous_content = content

Si el contenido actual del portapapeles es diferente del anterior, el programa imprime el contenido actual y actualiza previous_content a content. Esto configura el bucle para la siguiente vez que el usuario copie texto nuevo al portapapeles.

time.sleep(0.01) # Pausa para evitar el acaparamiento de CPU.

Si el contenido del portapapeles es el mismo de antes, podríamos hacer que el programa no haga nada. Sin embargo, este programa puede ejecutar el bucle fácilmente miles de veces por segundo y es poco probable que el usuario actualice el portapapeles con esa frecuencia. (Seguramente acabarían con las teclas **Control** y **C** de su teclado si lo intentasen). Para evitar que el programa acapare la CPU ejecutando un bucle improductivo lo más rápido posible, introducimos un retraso de 0,01 segundos para que el bucle busque

actualizaciones del portapapeles solo 100 veces por segundo.

except KeyboardInterrupt:

pass

La última parte del programa es la proposición except, que usa la declaración pass de Python. Esta declaración no hace nada literalmente, pero Python espera al menos una línea en el bloque que sigue a la declaración except. Para esto se creó la declaración pass. Cuando el usuario pulsa **Control-C**, la ejecución se mueve a esta proposición except y pasa al final del programa, donde termina.

Con esta *app* ejecutándose, el usuario puede pegar varias cosas sin tener que estar venga a cambiar entre aplicaciones. Pequeños programas como este pueden facilitar mucho tu flujo de trabajo, sobre todo si tienes que hacer esta tarea todos los días. Ahora vamos a desplegar el programa en los distintos sistemas operativos.

Windows

Guarda el archivo de Python como C:\Users\al\Scripts\cliprec.py (cambiando al por tu nombre de usuario). En la misma carpeta Scripts, crea un archivo cliprec.bat con el siguiente contenido:

```
@call %HOMEDRIVE%%HOMEPATH%\Scripts\.venv\Scripts  
\activate.bat
```

```
@python %HOMEDRIVE%%HOMEPATH%\Scripts\cliprec.py %*
```

```
@pause
```

```
@deactivate
```

Ya puedes ejecutar este programa desde el terminal o pulsando **Windows-R** para abrir el cuadro de diálogo Ejecutar e introduciendo cliprec.

macOS

Guarda el archivo de Python como `/Users/al/Scripts/cliprec.py` (cambiando al por tu nombre de usuario). En la misma carpeta Scripts, crea un archivo de texto con el nombre `cliprec.command` y el siguiente contenido:

```
source ~/Scripts/.venv/bin/activate
```

```
python3 ~/Scripts/cliprec.py
```

```
deactivate
```

A continuación, en un terminal, pasa con `cd` a la carpeta Scripts y ejecuta `chmod u+x cliprec.command`:

```
al@Als-MacBook-Pro ~ % cd ~/Scripts
```

```
al@Als-MacBook-Pro Scripts % chmod u+x cliprec.command
```

Ya puedes ejecutar este programa pulsando **Comando-Barra espaciadora** para abrir Spotlight e introduciendo `cliprec.command`.

Ubuntu Linux

Guarda el archivo de Python como `/home/al/Scripts/cliprec.py` (cambiando al por tu nombre de usuario). En la misma carpeta Scripts, crea un archivo de texto llamado `cliprec` con el siguiente contenido:

```
#!/usr/bin/env bash
```

```
source ~/Scripts/.venv/bin/activate
```

```
python3 ~/Scripts/cliprec.py
```

```
read -p "Press any key to continue..." -n1 -s
```

```
deactivate
```

A continuación, en un terminal, pasa con `cd` a la carpeta Scripts y ejecuta `chmod u+x cliprec`:

```
al@al-VirtualBox:~$ cd ~/Scripts
```

al@al-VirtualBox:~/Scripts\$ **chmod u+x cliprec**

Por último, crea un archivo de texto guardado como `~/.local/share/applications/cliprec.desktop` con el siguiente contenido:

[Desktop Entry]

Name = Clipboard Recorder

Exec = gnome-terminal -- /home/al/cliprec

Type = Application

Ya puedes ejecutar este programa pulsando la tecla **Meta** para abrir Dash y escribiendo Clipboard Recorder, o introduciendo los primeros caracteres del nombre y dejando que se autocomplete.

Compilar programas de Python con PyInstaller

Python se describe a menudo como un lenguaje interpretado, aunque los lenguajes de programación en sí ni se interpretan ni se compilan. Podemos crear un intérprete o un compilador para cualquier lenguaje. Los programas escritos en Python se ejecutan sobre todo por intérpretes, pero también es posible crear programas ejecutables a partir de código Python con el paquete PyInstaller, que genera ejecutables que se pueden ejecutar desde la línea de comandos.

PyInstaller no compila programas de Python en código máquina en sí, sino que crea un programa ejecutable que contiene una copia del intérprete de Python y el *script*. Por consiguiente, estos programas tienden a ser muy grandes. Incluso un sencillo programa “Hello, world” compilado con PyInstaller puede acercarse a los 8 MB de tamaño, que es, literalmente, mil veces más grande que la versión en lenguaje de ensamblador. Sin embargo, la ventaja de compilar tu programa de Python es que puedes compartirlo con otros que no tengan Python instalado. Podrías enviarles un archivo ejecutable.

Puedes instalar PyInstaller ejecutando `pip install pyinstaller`. Debes ejecutar PyInstaller con el sistema operativo en el que quieres que se ejecute. Es decir, si estás en Windows, PyInstaller puede crear un

ejecutable para Windows, pero no para macOS o Linux, y viceversa.

Desde el terminal, ejecuta el siguiente comando (con python3 en vez de python en macOS y Linux) para compilar un *script* de Python llamado `yourScript.py`:

```
C:\Users\al> python -m PyInstaller --onefile yourScript.py
```

```
378 INFO: PyInstaller: X.X.X
```

```
378 INFO: Python: 3.XX.XX
```

```
392 INFO: Platform: Windows-XX-XX.X.XXXX
```

```
393 INFO: wrote C:\Users\al\Desktop\hello-test\hello.spec
```

```
399 INFO: UPX is not available.
```

```
--fragmento omitido--
```

```
11940 INFO: Appending PKG archive to EXE
```

```
11950 INFO: Fixing EXE headers
```

```
13622 INFO: Building EXE from EXE-00.toc completed successfully.
```

Fíjate en que PyInstaller va con P e I mayúsculas, si no lo pones así recibirás un mensaje de error No module named pyinstaller. Observa también que el argumento `--onefile` lleva dos guiones.

Después de ejecutar PyInstaller, habrá una carpeta `build` (que puedes borrar) y una carpeta `dist`. Esta última contiene el programa ejecutable. No es necesario crear un entorno virtual. Puedes copiar el programa en otros ordenadores o enviarlo adjunto a un correo electrónico. Ten en cuenta que, como medida de seguridad, muchos proveedores de correo bloquean mensajes que contienen ejecutables.

Estas instrucciones funcionan para programas de Python básicos. La documentación en línea de <https://pyinstaller.org> ofrece más detalles.

Resumen

En este capítulo, has aprendido a sacar tus programas del editor de código y desplegarlos para que los usuarios puedan ejecutarlos cómodamente y con rapidez. También has descubierto más directrices sobre el diseño de programas que tienen interfaces basadas en texto en vez de interfaces gráficas modernas. Aunque una GUI es agradable para el usuario, las TUI son más fáciles de escribir. Cuando automatices tareas para ti, no merece la pena el esfuerzo extra para hacer que un programa parezca una aplicación profesional. Solo necesitas algo que funcione.

Dicho esto, hay varias maneras en las que puedes diseñar tu programa para que sea fácil de usar. A menudo, esto implica el terminal de línea de comandos, con el que muchos usuarios no están familiarizados. Puede llevar un tiempo aprender los conceptos de la línea de comandos, como navegar por el sistema de archivos con `cd` y `dir/ls`, la variable de entorno `PATH` y los argumentos de línea de comandos, pero el terminal permite emitir comandos y ejecutar programas muy deprisa, sobre todo cuando terminas de escribir y los despliegas.

Este capítulo también cubría varios paquetes de terceros. El paquete `Bext` permite añadir texto de colores, colocar el cursor y limpiar la pantalla. El paquete `PyMsgBox` crea cuadros de diálogo con GUI para alertas o entrada básica con la ventana de terminal. Como es posible que algún día ejecutes programas que requieren versiones incompatibles de un mismo paquete, es mejor ejecutar los *scripts* en entornos virtuales. Puedes crear entornos virtuales con el módulo `venv` que viene con Python. Los entornos virtuales se activan desde el terminal y pueden evitar que se estropeen programas existentes dándonos un espacio separado para instalar paquetes.

Por último, el paquete `PyInstaller` permite compilar los archivos `.py` en programas ejecutables. Pueden tener un tamaño de varios megabytes, pero es una forma de compartir programas con compañeros que podrían no tener Python (y los paquetes de terceros que use tu programa) en su ordenador.

En este capítulo no hemos hablado mucho de conceptos de programación, sino más bien de cómo hacer que los programas sean

usables y cómodos en el día a día. Llegados a este punto, ya sabes suficiente sintaxis de Python para crear programas básicos (¡aunque siempre hay algo más que aprender!). En esta segunda parte del libro, exploraremos muchos paquetes de terceros que amplían las capacidades de nuestros programas de Python.

Preguntas prácticas

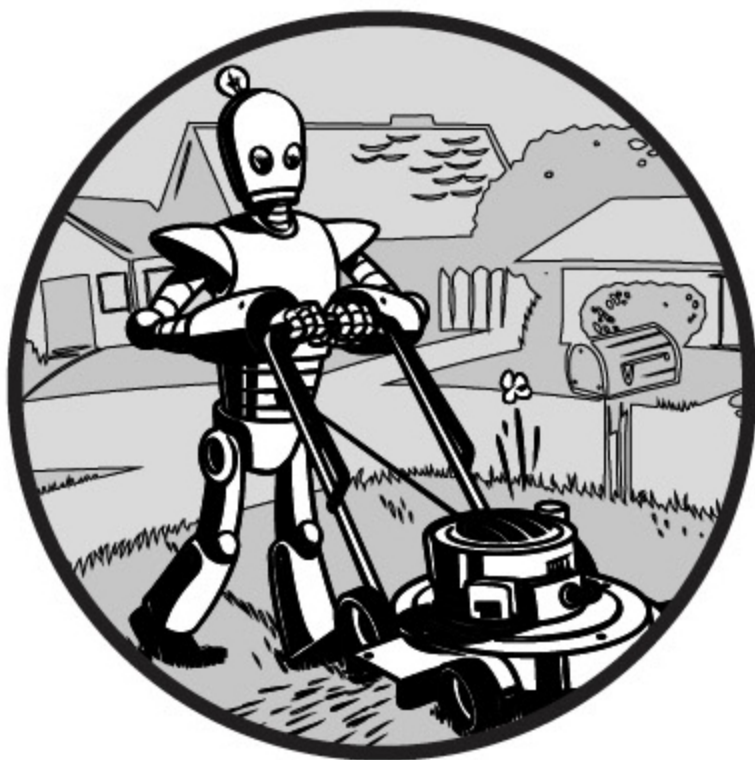
1. ¿Qué comando hace una lista con el contenido de una carpeta en Windows? ¿Y en macOS y Linux?
2. ¿Qué contiene la variable de entorno PATH?
3. ¿Qué contiene la variable `_file_`?
4. ¿Qué comando borra el texto de la ventana de terminal en Windows? ¿Y en macOS y Linux?
5. ¿Cómo se crea un entorno virtual nuevo?
6. ¿Qué argumento de línea de comandos habría que pasar a PyInstaller al compilar programas?

Programa de práctica: haz tus programas desplegables

Haz tus programas existentes más fáciles de ejecutar creando *scripts* de en una carpeta PATH que los ejecute o compilándolos con PyInstaller. Haz esto para los siguientes proyectos:

- “Hacer una copia de seguridad de una carpeta en un archivo ZIP”, del capítulo 11.
- “Extraer información de contacto de documentos grandes”, del capítulo 9.
- “Añadir viñetas a marcado wiki”, del capítulo 8.
- “Simulador de un tablero de ajedrez interactivo”, del capítulo 7.
- Cualquier otro programa que hayas creado y quieras iniciar fácilmente o compartir con otros.

Extraer información de sitios web



En esos raros y aterradores momentos en los que estoy sin wifi, me doy cuenta de hasta qué punto lo que hago en el ordenador es en realidad lo que hago en Internet. Por costumbre, paso el tiempo comprobando el correo electrónico, las redes sociales o preguntándome cosas como: “¿Tuvo Kurtwood Smith papeles protagonistas antes de aparecer en el *RoboCop* original de 1987?”.¹

Como buena parte del trabajo con un ordenador implica usar Internet, sería genial que tus programas pudieran conectarse. El raspado web, o *web scraping*, es un término que designa el uso de un programa para descargar y procesar contenido de la red. Por

ejemplo, Google ejecuta muchos programas de estos para indexar páginas web para su motor de búsqueda. En este capítulo, aprenderás sobre los siguientes módulos, que facilitan la extracción de información de páginas web con Python:

- webbrowser: Viene con Python y abre un navegador con una página específica.
- requests: Descarga archivos y páginas web de Internet.
- Beautiful Soup (bs4): Analiza el HTML, el formato en el que están escritas las páginas web, para extraer la información que queremos.
- Selenium: Abre y controla un navegador web para, por ejemplo, rellenar formularios y simular clics de ratón.
- Playwright: Abre y controla un navegador web. Es más reciente que Selenium y tiene funciones adicionales.

HTTP y HTTPS

Cuando visitamos un sitio web, su dirección, como <https://author.com/example3.html>, se conoce como URL (*Uniform Resource Locator*, localizador uniforme de recursos). Las letras HTTPS en la URL son de *HyperText Transfer Protocol Secure* (protocolo de transferencia de hipertexto seguro), el protocolo que emplea el navegador para acceder a los sitios web. Los paquetes de este capítulo permiten a nuestros *scripts* acceder a servidores web mediante este protocolo.

Para ser exactos, HTTPS es una versión encriptada de HTTP, que protege nuestra privacidad mientras usamos Internet. Si estuviésemos usando HTTP, los ladrones de identidad, las agencias de seguridad nacional y nuestro proveedor de servicios de Internet podrían ver el contenido de las páginas que hemos visitado, incluidas las contraseñas o los números de tarjeta de crédito que introduzcamos. Utilizar una red virtual privada (VPN, *Virtual Private Network*) podría impedir que tu proveedor de servicios de Internet vea tu tráfico en la red, pero podría verlo el proveedor de la VPN. Un proveedor de VPN sin escrúpulos podría vender la información sobre qué sitios visitas a un bróker de información. (Tom Scott

explica lo que ofrece y no ofrece una VPN en su vídeo *This Video Is Sponsored by VPN*).

En cambio, cualquier contenido que veas con HTTPS estará encriptado y oculto. Los sitios web solían reservar el HTTPS para páginas que enviaban contraseñas y números de tarjeta, pero actualmente la mayoría de los sitios encriptan todo el tráfico. Ten en cuenta, no obstante, que la identidad del sitio web que visitas puede seguir sabiéndose; nadie podrá ver exactamente qué te has descargado de CatPhotos.com, pero verán que te has conectado a ese sitio web y suponer que estabas viendo fotos de gatos. El navegador Tor, que utiliza la red de anonimización Tor, ofrece una auténtica navegación anónima. Puedes descargarlo en <https://www.torproject.org/download/>.

Proyecto 6: Ejecutar un programa con el módulo webbrowser

Vamos a descubrir el módulo webbrowser de Python usándolo en un proyecto de programación. Su función open() puede abrir un navegador con una URL especificada. Introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > import webbrowser
```

```
> > > webbrowser.open('https://inventwithpython.com/')
```

Una pestaña del navegador se abrirá con la URL <https://inventwithpython.com>. Esto es casi lo único que puede hacer el módulo webbrowser. Aun así, la función open() hace posibles algunas cosas interesantes.

Por ejemplo, es una pesadez copiar una dirección en el portapapeles y pegarla en el navegador cada vez que quieres verla en un mapa en OpenStreetMap. Podrías eliminar unos cuantos pasos de esa tarea escribiendo un guion sencillo que abra el mapa en el navegador con el contenido del portapapeles. De este modo, solo habría que copiar la dirección al portapapeles y ejecutar el *script* para que se cargue el mapa. Podemos poner la dirección directamente en la URL de OpenStreetMap, así que solo necesitamos la función webbrowser.open().

Esto es lo que hace el programa:

- Coge una dirección postal de los argumentos de línea de comandos o del portapapeles.
- Abre el navegador web con la página OpenStreetMap de esa dirección.

Esto significa que el código tiene que hacer lo siguiente:

- Leer los argumentos de la línea de comandos desde `sys.argv`.
- Leer el contenido del portapapeles.
- Llamar a la función `webbrowser.open()` para abrir el navegador web.

Abre una nueva pestaña en el editor de archivos y guarda como `showmap.py`.

Paso 1: descubrir la URL

Siguiendo las instrucciones del capítulo 12, configura un archivo `showmap.py` para que, cuando lo ejecutes desde la línea de comandos, por ejemplo con:

```
C:\Users\al> showmap 777 Valencia St, San Francisco, CA 94110
```

el *script* use los argumentos de línea de comandos en vez del portapapeles. Si no hay argumentos de línea de comandos, el programa sabrá usar el contenido del portapapeles.

Para hacer esto, necesitas descubrir la URL de una dirección determinada. Al cargar <https://www.openstreetmap.org> en el navegador y buscar una dirección, la URL de la barra de direcciones es algo como: <https://www.openstreetmap.org/search?query=777%20Valencia%20St%2C%20San%20Francisco%2C%20CA%2094110#map=19/37.75897/-122.42142>.

Puedes probar si la URL necesita o no la parte `#map` quitándola de la barra y visitando el sitio para ver si se carga bien. Así, tu

programa puede configurarse para abrir un navegador en [https://www.openstreetmap.org/search?query= <la_cadena_de_tu_dirección>](https://www.openstreetmap.org/search?query=<la_cadena_de_tu_dirección>) (donde = <la_cadena_de_tu_dirección> es la dirección que quieres ver en el mapa). Observa que tu navegador maneja automáticamente cualquier codificación de URL necesaria, como convertir los caracteres en blanco de la URL en %20.

Paso 2: manejar los argumentos de línea de comandos

Usa este código:

```
# showmap.py – Abre un mapa en el navegador usando una dirección de la
```

```
# línea de comandos o el portapapeles
```

```
import webbrowser, sys
```

```
if len(sys.argv) > 1:
```

```
# Obtiene la dirección de la línea de comandos.
```

```
address = ' '.join(sys.argv[1:])
```

```
# PENDIENTE: Obtener la dirección del portapapeles.
```

```
# PENDIENTE: Abrir el navegador web.
```

Primero, hay que importar el módulo webbrowser para abrir el navegador y el módulo sys para leer los posibles argumentos de línea de comandos. La variable sys.argv almacena el nombre de archivo y los argumentos de línea de comandos como una lista. Si esta lista tiene algo más que el nombre del archivo, len(sys.argv) se evalúa como un entero mayor que 1, lo que significa que hay argumentos de línea de comandos.

Estos argumentos suelen ir separados por espacios, pero, en este caso, queremos que se interpreten todos como una sola cadena. Como sys.argv es una lista de cadenas, podemos pasársela al método join(), que devolverá un solo valor de cadena. No queremos el nombre del programa en esta cadena, así que pasamos sys.argv[1:

en vez de `sys.argv` para quitar el primer elemento de la matriz. La cadena final a la que se evalúa esta expresión se guarda en la variable `address`.

Si ejecutamos el programa introduciendo esto en la línea de comandos:

```
showmap 777 Valencia St, San Francisco, CA 94110
```

la variable `sys.argv` contendrá este valor de lista:

```
['showmap.py', '777', 'Valencia', 'St', ' ', 'San', 'Francisco', ' ', 'CA', '94110']
```

Después de unir `sys.argv[1:` con un carácter de espacio, la variable `address` contendrá la cadena `'777 Valencia St, San Francisco, CA 94110'`.

Paso 3: recuperar el contenido del portapapeles

Para buscar la URL en el portapapeles, necesitas el siguiente código:

```
# showmap.py - Abre un mapa en el navegador usando una dirección de la
```

```
# línea de comandos o el portapapeles
```

```
import webbrowser, sys, pyperclip
```

```
if len(sys.argv) > 1:
```

```
# Obtiene la dirección de la línea de comandos.
```

```
address = ' '.join(sys.argv[1:])
```

```
else:
```

```
# Obtiene la dirección del portapapeles.
```

```
address = pyperclip.paste()
```

```
# Abre el navegador web.
```

`webbrowser.open('https://www.openstreetmap.org/search?query=' + address)`

Si no hay argumentos de línea de comandos, el programa asumirá que la dirección está en el portapapeles. Podemos coger el contenido del portapapeles con `pyperclip.paste()` y guardarlo en una variable `address`. Por último, para abrir un navegador con la URL de OpenStreetMap, llamamos a `webbrowser.open()`.

Aunque algunos de los programas que escribas harán tareas enormes que te ahorrarán varias horas, puede ser igual de satisfactorio usar un programa que te ahorre unos segundos cada vez que haces una tarea habitual, como sacar un mapa de una dirección. La tabla 13.1 compara los pasos necesarios para ver el mapa con y sin `showmap.py`.

Tabla 13.1. Obtener un mapa con y sin `showmap.py`.

Obtener un mapa manualmente

1. Seleccionar la dirección.
2. Copiar la dirección.
3. Ejecutar `showmap.py`.
4. Ir a <https://www.openstreetmap.org>
5. Hacer clic en el campo para direcciones.
6. Pegar la dirección.
7. Pulsar **Intro**.

Tenemos suerte de que el sitio web OpenStreetMap no requiere ninguna interacción para mostrar el mapa, podemos simplemente poner la dirección en una URL. El *script* `showmap.py` hace esta tarea menos tediosa, sobre todo si la realizas con frecuencia.

Ideas para programas similares

Siempre y cuando tengamos una URL, el módulo `webbrowser` permite al usuario saltarse el paso de abrir el navegador y entrar a un sitio web. Otros programas podrían aprovechar esta funcionalidad para hacer lo siguiente:

- Abrir todos los enlaces de una página en distintas pestañas del navegador.

- Abrir el navegador en la URL del sitio meteorológico local.
- Abrir varios sitios de redes sociales o con marcadores que visites habitualmente.
- Abrir un archivo .html local en tu disco duro.

La última sugerencia es útil para mostrar archivos de ayuda. Aunque tu programa podría usar `print()` para mostrar una página de ayuda al usuario, llamar a `webbrowser.open()` para abrir un archivo .html con información de ayuda permite que la página tenga distintas fuentes, colores, tablas e imágenes. En vez de usar el prefijo `https://`, usa `file://`. Por ejemplo, tu carpeta Escritorio debería tener un archivo `help.html` local en `file:///C:/Users/al/Desktop/help.html` en Windows o `file:///Users/al/Desktop/help.html` en macOS.

Descargar archivos de la web con el módulo requests

El módulo `requests` permite descargar fácilmente archivos desde la web sin tener que preocuparse de cuestiones complicadas como los errores de red, el enrutamiento de la conexión y la compresión de datos. Este módulo no viene con Python, así que tendrás que instalarlo antes de poder usarlo, siguiendo las instrucciones del apéndice A.

Descargar páginas web

La función `requests.get()` toma una cadena que representa una URL para descargar. Llamando `type()` con el valor de retorno de la función, vemos que devuelve un objeto `Response`, que contiene la respuesta que el servidor web ha dado a nuestra solicitud. Explicaré el objeto `Response` con más detalle después, por ahora escribe esto en el intérprete de comandos interactivo con tu ordenador conectado a Internet:

```
> > > import requests
```

```
1 > > > response = requests.get('https://
  automatetheboringstuff.com/files/rj.txt')
```

```
> > > type(response)
```

```
<class 'requests.models.Response'>
```

```
2 > > > response.status_code == requests.codes.ok
```

```
True
```

```
> > > len(response.text)
```

```
178978
```

```
> > > print(response.text[:210])
```

The Project Gutenberg EBook of Romeo and Juliet, by William Shakespeare

This eBook is for the use of anyone anywhere at no cost and with almost no restrictions whatsoever. You may copy it, give it away or

La URL nos lleva a una página web que contiene el texto completo de *Romeo y Julieta* en inglés 1. Sabemos que la solicitud de la página ha ido bien comprobando el atributo `status_code` del objeto `Response`. Si es igual que el valor `requests.codes.ok`, todo ha ido bien 2. (Por cierto, el código de estado para “OK” en HTTP es 200. Puede que estés ya familiarizado con el código de estado 404 de “No encontrado”).

Si la solicitud ha ido bien, la página descargada se guarda como una cadena en la variable `text` del objeto `Response`. Esta gran cadena contiene la obra entera; la llamada a `len(response.text)` muestra que tiene más de 178.000 caracteres. Por último, llamar a `print(response.text[:210])` muestra los primeros 210 caracteres.

Si la solicitud falla y aparece un mensaje de error, como `Failed to establish a new connection` o `Max retries exceeded`, comprueba tu conexión a Internet. Conectarse a servidores puede ser bastante complicado y no puedo darte aquí una lista con todos los problemas potenciales. Puedes encontrar las causas más habituales del error buscando en la web el mensaje entrecomillado. Ten en cuenta también que, si descargas una página web con `requests`, solo

obtendrás el contenido HTML de la página; tendrás que descargar las imágenes y otros objetos multimedia por separado.

Buscar errores

Como acabamos de ver, el objeto Response tiene un atributo `status_code` que podemos comprobar con `requests.codes.ok` para ver si la descarga ha tenido éxito. Una manera más fácil de comprobarlo es llamar al método `raise_for_status()` del objeto Response. Este método lanzará una excepción si se produce un error mientras se descarga el archivo y no hará nada si la descarga va bien. Introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > response = requests.get('https://  
inventwithpython.com/page_that_does_not_exist')  
  
> > > response.raise_for_status()
```

Traceback (most recent call last):

File "<python-input-0>", line 1, in <module>

File "C:\Users\Al\AppData\Local\Programs\Python\PythonXX\lib
\site-packages\
requests\models.py", line 940, in raise_for_status

raise HTTPError(http_error_msg, response = self)

requests.exceptions.HTTPError: 404 Client Error: Not Found for url:

https://inventwithpython.com/page_that_does_not_exist.html

El método `raise_for_status()` es una forma fácil de asegurarse de que un programa se detiene si la descarga no va bien. Por lo general, conviene que el programa se detenga en cuanto se produzca un error inesperado. Si una descarga fallida no es para tanto, puedes poner la línea `raise_for_status()` entre declaraciones `try` y `except` para manejar el error sin parar el programa:

```
import requests
```

```
response = requests.get('https://inventwithpython.com/  
page_that_does_not_exist')
```

try:

```
response.raise_for_status()
```

except Exception as exc:

```
print(f'There was a problem: {exc}')
```

Esta llamada a `raise_for_status()` hace que el programa muestre lo siguiente:

There was a problem: 404 Client Error: Not Found for url:

https://inventwithpython.com/page_that_does_not_exist.html

Llama siempre a `raise_for_status()` después de llamar a `requests.get()`. Deberías asegurarte de que la descarga ha funcionado realmente antes de que continúe el programa.

Guardar archivos descargados en el disco duro

Desde aquí, puedes guardar la página web en un archivo de tu disco duro con la función `open()` estándar y el método `write()`. Sin embargo, debes abrir el archivo en modo de escritura binario pasando la cadena `'wb'` como segundo argumento a `open()`. Aunque la página sea texto simple (como el texto de *Romeo y Julieta* que hemos descargado antes), hay que escribir datos binarios en vez de datos de texto para mantener la codificación Unicode del texto.

Para escribir la página web en un archivo, puedes usar un bucle `for` con el método `iter_content()` del objeto `Response`:

```
> > > import requests
```

```
> > > response = requests.get('https://  
automatetheboringstuff.com/files/rj.txt')
```

```
> > > response.raise_for_status()
```

> > > **with open('RomeoAndJuliet.txt', 'wb') as play_file:**

... for chunk in response.iter_content(100000):

. play_file.write(chunk)

.

100000

78978

El método `iter_content()` devuelve “trozos” del contenido en cada iteración por el bucle. Cada trozo es del tipo de datos bytes y podemos especificar cuántos bytes contendrá cada trozo. Cien mil bytes suele ser un buen tamaño, así que pasamos 100000 como argumento a `iter_content()`.

Ya tienes el archivo `RomeoAndJuliet.txt` en el directorio de trabajo actual. Observa que, aunque el nombre del archivo en el sitio web era `rj.txt`, el nombre en el disco duro es distinto.

El método `write()` devuelve el número de bytes escritos en el archivo. En el ejemplo anterior, había 100.000 en el primer trozo y el resto del archivo necesitaba solo 78.978 bytes.

Un repaso de la descarga y el guardado de archivos

Para repasar, aquí tienes el proceso completo para descargar y guardar un archivo:

- Llamar a `requests.get()` para descargar el archivo.
- Llamar a `open()` con `'wb'` para crear un nuevo archivo en modo binario.
- Pasar en bucle por el método `iter_content()` del objeto `Response`.
- Llamar a `write()` en cada iteración para escribir el contenido en el archivo.

¡Eso es lo más importante del módulo `requests`! Puedes aprender

más sobre las demás características del módulo en <https://requests.readthedocs.io/en/latest/>.

Si quieres descargar archivos de vídeo de sitios como YouTube, Facebook, X (anteriormente Twitter), usa el módulo yt-dlp, que veremos en el capítulo 24.

Acceder a una API del tiempo

Las *apps* que utilizamos están diseñadas para interactuar con usuarios humanos. Sin embargo, podemos escribir programas para que interactúen con otros programas a través de su interfaz de programación de aplicaciones (API, *Application Programming Interface*), que es la especificación que define cómo un software (como nuestros programas de Python) puede comunicarse con otros softwares (como el servidor web de un sitio de meteorología). Los servicios en línea suelen tener API. Por ejemplo, podríamos escribir un *script* de Python para publicar en nuestras cuentas en redes sociales o descargar fotos nuevas. En este apartado, escribiremos un *script* que accede a información meteorológica del sitio web gratuito OpenWeather.

Casi todos los servicios en línea requieren registrar una dirección de correo electrónico para usar su API. Incluso aunque la API sea gratuita, puede haber límites en la cantidad de solicitudes que se le hagan al día o a la hora. Si te preocupa recibir correos basura, puedes usar una dirección desechable temporal como <https://10minutemail.com>. Ten en cuenta que deberías usar ese tipo de servicios solo para registrar cuentas que no te importen, ya que un servicio de correo electrónico sin escrúpulos podría hacerse con el control de tu cuenta restableciendo la contraseña en tu nombre.

Para empezar, crea una cuenta gratuita en <https://openweathermap.org>. Las cuentas gratuitas están limitadas a 60 solicitudes por minuto. Es más que suficiente para proyectos de programación pequeños o medianos. Si tu programa necesita más (por ejemplo, porque procesa las solicitudes de cientos de visitantes simultáneos en tu sitio web), puedes pasarte a un plan de pago. Los servicios en línea te dan una clave API, que es una especie de

contraseña que identifica tu cuenta en tus solicitudes a la API. ¡Mantén la clave API en secreto! Cualquiera con la clave puede hacer solicitudes a la API como si fueran desde tu cuenta. Si escribes un programa que utiliza una clave API, considera hacer que el programa lea un archivo de texto que contenga la clave en vez de incluirla directamente en el código fuente. De ese modo, puedes compartir tu programa con otros (que pueden registrarse para recibir su propia clave API) sin preocuparte por exceder el límite de solicitudes de tu cuenta.

Muchas API HTTP envían su respuesta como una cadena larga. Esta cadena suele estar en formato JSON o XML. En el capítulo 18 veremos estos formatos con más detalle, de momento basta con que sepas que `json.loads(response.text)` devuelve una estructura de datos de Python de listas y diccionarios que contienen los datos JSON en `response.text`. Los ejemplos de este capítulo guardan los datos en una variable llamada `response_data`, pero es una elección arbitraria, puedes usar el nombre de variable que quieras.

Todos los servicios en línea ofrecen documentación sobre cómo usar su API. La de OpenWeather está en <https://openweathermap.org/api>. Una vez que tienes la cuenta y la clave API de la página My API keys de https://home.openweathermap.org/api_keys, úsala en el siguiente código con el intérprete de comandos interactivo. Yo usaré ‘30ee784a80d81480dab1749d33980112’ como clave de API falsa en este ejemplo, así que no la utilices en tu código porque no funcionará. Primero usaremos OpenWeather para encontrar la latitud y longitud de San Francisco:

```
> > > import requests

> > > city_name = 'San Francisco'

> > > state_code = 'CA'

> > > country_code = 'US'

> > > API_key = '30ee784a80d81480dab1749d33980112' #
Not a real API key

> > > response = requests.get(f'https://
```

api.openweathermap.org/geo/1.0/

```
direct?q={city_name},{state_code},{country_code}
&appid={API_key}'
```

```
> > > response.text # Esto es una cadena de Python.
```

```
{["name": "San Francisco", "local_names": {"id": "San Francisco",
```

```
--fragmento omitido--
```

```
, "lat": 37.7790262, "lon": -122.419906, "country": "US", "state": "California"]]
```

```
> > > import json
```

```
> > > response_data = json.loads(response.text)
```

```
> > > response_data # Esto es una estructura de datos de Python.
```

```
[{"name": "San Francisco", "local_names": {"id": "San Francisco",
```

```
--fragmento omitido--
```

```
, "lat": 37.7790262, "lon": -122.419906, "country": "US", "state": "California"]]
```

Para entender los datos de la respuesta, deberías consultar la documentación en línea para la API de OpenWeather o examinar el diccionario de `response_data` en el intérprete de comandos interactivo. Descubrirás que la respuesta es una lista cuyo primer elemento (en el índice 0) es un diccionario con claves 'lat' y 'lon'. Los valores para estas claves son valores flotantes de latitud y longitud:

```
> > > response_data[0]['lat']
```

```
37.7790262
```

```
> > > response_data[0]['lon']
```

```
-122.419906
```

La URL específica que se usa para hacer una solicitud a una API se llama *endpoint*. Las cadenas formateadas de este ejemplo

reemplazan las partes entre llaves con los valores variables. La parte `direct?q={city_name},{state_code},{country_code}&appid={API_key}` del ejemplo anterior se convierte en `direct?q=San Francisco,CA,US&appid=30ee784a80d81480dab1749d33980112`.

A continuación, puedes usar esta información de latitud y longitud para hallar la temperatura actual en San Francisco:

```
> > > lat = json.loads(response.text)[0]['lat']
```

```
> > > lon = json.loads(response.text)[0]['lon']
```

```
> > > response = requests.get(f'https://api.openweathermap.org/data/2.5/
```

```
weather?lat={lat}&lon={lon}&appid={API_key}')
```

```
> > > response_data = json.loads(response.text)
```

```
> > > response_data
```

```
{'coord': {'lon': -122.4199, 'lat': 37.779}, 'weather': [{'id': 803,
```

```
--fragmento omitido--
```

```
'timezone': -25200, 'id': 5391959, 'name': 'San Francisco', 'cod': 200}
```

```
> > > response_data['main']['temp']
```

```
285.44
```

```
> > > round(285.44 - 273.15, 1) # Convierte Kelvin a Celsius.
```

```
12.3
```

```
> > > round(285.44 * (9 / 5) - 459.67, 1) # Convierte Kelvin a Fahrenheit.
```

```
54.1
```

Observa que OpenWeather devuelve la temperatura en Kelvin, así

que habrá que hacer algunos cálculos para pasarla a Celsius o Fahrenheit. Vamos a descomponer la URL completa del *endpoint* de geolocalización del ejemplo anterior:

- `https://`: El esquema usado para acceder al servidor, que es el nombre del protocolo (casi siempre HTTPS para API en línea) seguido de dos puntos y dos barras invertidas.
- `api.openweathermap.org`: El nombre de dominio del servidor web que maneja la solicitud a la API.
- `/geo/1.0/direct`: La ruta de la API.

•`?q={city_name},{state_code},{country_code}&appid={API_key}`: La cadena de consulta de la URL. Hay que reemplazar lo que hay entre llaves por valores reales; piensa en ellos como los parámetros de una llamada a función. En la codificación URL, el nombre del parámetro y el valor del argumento están separados por un `&`

Puedes coger la URL del *endpoint* (con la cadena de consulta completa) y pegarla en tu navegador para ver el texto de respuesta directamente. Esto es una buena práctica cuando estás aprendiendo a usar una API. El texto de respuesta para API basadas en web suele estar en formato JSON o XML.

Para evitar confusiones al actualizar una API, la mayoría de los servicios en línea incluyen un número de versión como parte de la URL. Con el tiempo, el servicio puede sacar nuevas versiones de la API y desechar versiones anteriores. Llegado el caso, tendrás que actualizar el código de tus *scripts* para poder seguir usándolos.

La versión gratuita de OpenWeather ofrece previsiones a cinco días e información sobre precipitaciones, viento y contaminación del aire. Las páginas de documentación muestran a qué URL acceder para obtener estos datos, además de la estructura de la respuesta JSON para estas llamadas a la API. El código de los siguientes apartados asume que has ejecutado `response_data = json.loads(response.text)` para convertir el texto devuelto por el sitio web en una estructura de datos de Python.

Solicitar una latitud y una longitud

El *endpoint* para obtener las coordenadas de latitud y longitud de una ciudad es https://api.openweathermap.org/geo/1.0/direct?q={city_name},{state_code},{country_code}&appid={API_key}. El código de estado se refiere a la abreviatura del estado y solo se aplica a ciudades de Estados Unidos. El código de país es el código ISO 3166 de dos o tres letras, recogido en https://en.wikipedia.org/wiki/List_of_ISO_3166_country_codes. Por ejemplo, usa el código 'US' para Estados Unidos o 'NZ' para Nueva Zelanda. Tras convertir el texto JSON de respuesta en una estructura de datos de Python en una variable llamada `response_data`, puedes recuperar la siguiente información:

- `response_data[0]['lat']`: Contiene los grados de latitud de la ciudad como valor flotante.
- `response_data[0]['lon']`: Contiene los grados de longitud de la ciudad como valor flotante.

Si el nombre de la ciudad coincide con varias respuestas, la lista de `response_data` contendrá diccionarios en `response_data[0]`, `response_data[1]`, etc. Si OpenWeather es incapaz de localizar la ciudad, `response_data` será una lista vacía.

Buscar el tiempo actual

El *endpoint* para obtener la información meteorológica actual basándose en una latitud y una longitud es https://api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon}&appid={API_key}. Una vez convertido el texto JSON de respuesta en una estructura de datos de Python en una variable llamada `response_data`, se puede recuperar la siguiente información:

- `response_data['weather'][0]['main']`: Alberga una descripción como cadena, por ejemplo, 'Clear', 'Rain' o 'Snow'.
- `response_data['weather'][0]['description']`: Contiene una cadena más descriptiva, como 'light rain', 'moderate rain' o 'extreme rain'.
- `response_data['main']['temp']`: La temperatura actual en Kelvin.
- `response_data['main']['feels_like']`: La sensación térmica en Kelvin.

- `response_data['main']['humidity']`: El porcentaje de humedad.

Si se ha suministrado una latitud o una longitud incorrecta como argumento, `response_data` será un diccionario como `{"cod": "400", "message": "wrong latitude"}`.

Obtener una predicción meteorológica

El *endpoint* para obtener una previsión a cinco días basándose en una latitud y una longitud es `https://api.openweathermap.org/data/2.5/forecast?lat={lat}&lon={lon}&appid={API_key}`. Una vez convertido el texto JSON de respuesta en una estructura de datos de Python en una variable llamada `response_data`, puedes recuperar la siguiente información:

- `response_data['list']`: Tiene una lista de diccionarios que contienen la predicción meteorológica para un periodo dado.

- `response_data['list'][0]['dt']`: Alberga un sello de tiempo en forma de flotante Unix de *epoch*. Pasa este valor como argumento a `datetime.datetime.fromtimestamp()` para obtener el sello como objeto `datetime object`. El capítulo 19 explica el módulo `datetime` de Python con más detenimiento.

- `response_data['list'][0]['main']`: Guarda un diccionario con claves como `'temp'`, `'feels_like'`, `'humidity'`, entre otras.

- `response_data['list'][0]['weather'][0]`: Contiene un diccionario de descripciones con claves como `'main'`, `'description'`, etc.

La lista de `response_data['list']` incluye 40 diccionarios con predicciones en incrementos de tres horas para los próximos cinco días, aunque esto puede cambiar en futuras versiones de la API.

Explorar las API

Otros sitios web, como <https://weather.gov> y <https://www.weatherapi.com/>, ofrecen sus propias APIs del tiempo gratuitas. Cada API funciona a su manera, pero se suele acceder a ellas con solicitudes por HTTPS, en cuyo caso, podemos usar la

librería Requests y las respuestas devueltas en formato de texto JSON o XML. Sin embargo, es posible que alguien haya creado un paquete de terceros para facilitar el uso de estas APIs, con funciones de Python que gestionen el acceso a los *endpoints* y el análisis de las respuestas. Puedes encontrar paquetes así en <https://pypi.org>; lee la documentación del paquete para saber cómo se usa.

Entender HTML

Antes de trabajar con páginas web, debes aprender lo básico del lenguaje de marcado de hipertexto (HTML, *HyperText Markup Language*). HTML es el formato en el que se escriben los sitios web, mientras que las hojas de estilo (CSS, *Cascading Style Sheets*) son una forma de hacer cambios categóricos en el aspecto de los elementos HTML de una página web. Este capítulo asume que tienes cierta experiencia básica con HTML, pero, si necesitas un tutorial para iniciarte, te sugiero los siguientes sitios:

- https://developer.mozilla.org/es/docs/Learn_web_development/Core/Structuring_content.
- <https://www.freecodecamp.org/news/html-coding-introduction-course-for-beginners>.
- <https://www.khanacademy.org/computing/computer-programming/html-css>.

En este apartado, también aprenderás a acceder a las potentes herramientas de desarrollo de tu navegador, que facilitan mucho la extracción de información de Internet.

Explorar el formato

Un archivo HTML es un archivo de texto simple con la extensión .html. El texto en estos archivos está entre etiquetas HTML, que son palabras entre paréntesis angulares (< >). Las etiquetas indican al navegador cómo dar formato a la página web. Una etiqueta de apertura y una etiqueta de cierre pueden marcar un texto para crear un elemento HTML. El texto que se muestra es el contenido entre la etiqueta de apertura y la de cierre. Por ejemplo, el siguiente HTML

mostrará “Hello, world!” en el navegador, con “Hello” en negrita:

`<b>Hello</b>, world!`

En un navegador, este HTML queda como en la figura 13.1.

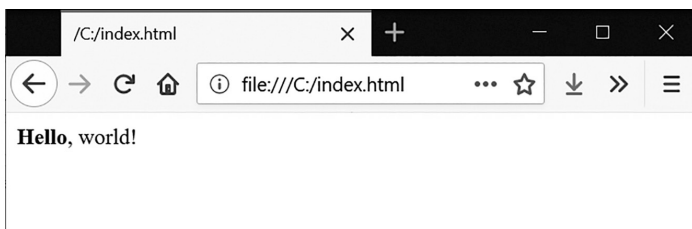


Figura 13.1. Hello, world! mostrado en el navegador.

La etiqueta de apertura `<b>` indica que el texto siguiente irá en negrita. La etiqueta de cierre `</b>` dice al navegador dónde termina el texto en negrita. Juntas, forman un elemento:

`<b>Hello</b>.`

Hay muchas etiquetas diferentes en HTML. Algunas tienen propiedades adicionales en forma de atributos dentro de los paréntesis angulares. Por ejemplo, la etiqueta `<a>` se aplica a un texto que debería ser un enlace y el atributo `href` determina la URL a la que se enlaza. Aquí tienes un ejemplo:

`<a href="https://inventwithpython.com">This text is a link</a>`

Algunos elementos tienen un atributo `id` que sirve para identificar de forma única el elemento en la página. A menudo, dirás a tus programas que busquen un elemento por su atributo `id`, así que buscarlo con las herramientas de desarrollo del navegador es una tarea habitual al escribir programas de raspado web.

Ver el código fuente de una página web

Tendrás que ver el HTML de las páginas web con las que trabajen tus programas, que es lo que se denomina código fuente. Para ello, haz clic con el botón secundario (o **Control-clic** en macOS) en cualquier parte de la página web y selecciona Ver código fuente de

la página (figura 13.2). El código fuente es el texto que recibe en realidad el navegador, que sabe cómo mostrar o representar la página web a partir de ese HTML.

Prueba a ver el HTML del código fuente de algunos de tus sitios favoritos; no pasa nada si no entiendes todo lo que estás viendo. No necesitas un máster en HTML para escribir programas sencillos de raspado web, te basta con saber lo justo para sacar datos de un sitio web existente.

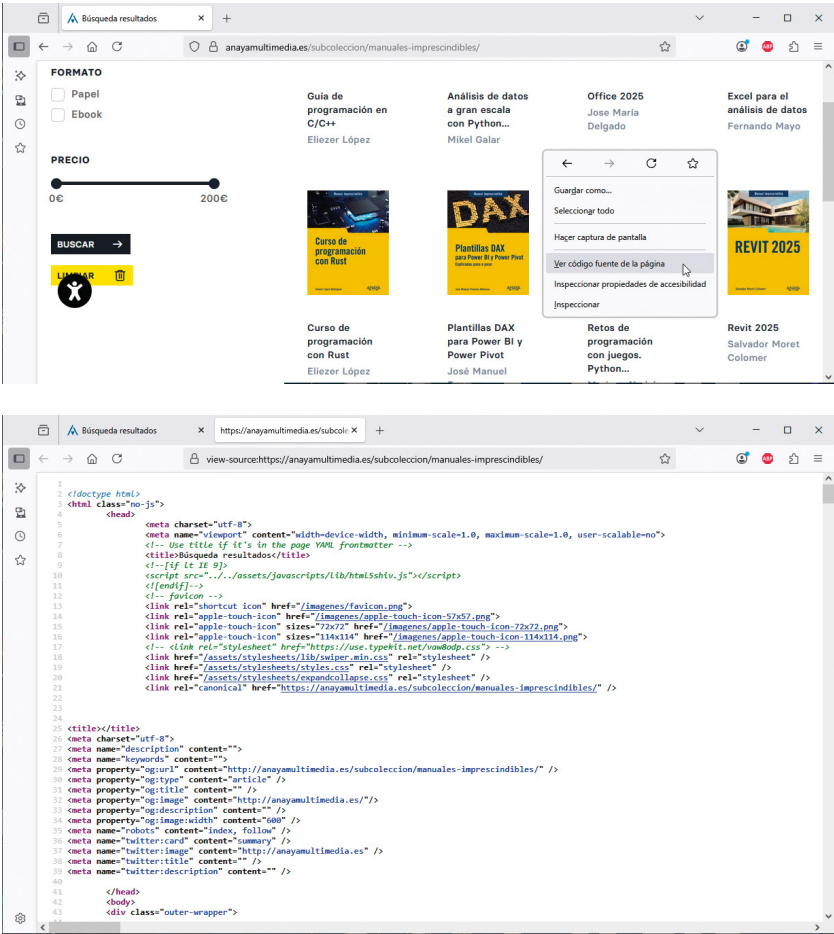


Figura 13.2. Vista del código fuente de una página.

Abrir las herramientas para desarrolladores de tu navegador

Además de ver el código fuente de una página web, puedes echar un vistazo al HTML con las herramientas para desarrolladores de tu navegador. En Firefox, Chrome y Microsoft Edge, pulsa **F12** para que aparezcan (figura 13.3). Pulsa de nuevo **F12** para hacerlas desaparecer.

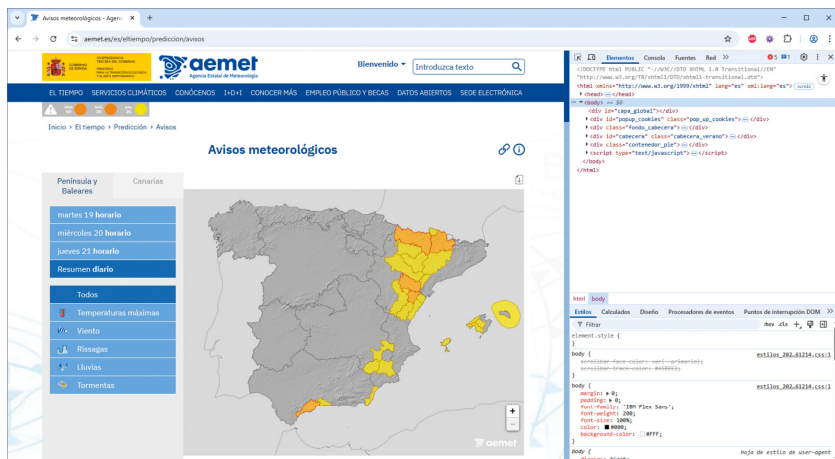


Figura 13.3. La ventana de herramientas para desarrolladores en el navegador Chrome.

Haz clic con el botón secundario en cualquier parte de la página web y selecciona Inspeccionar en el menú contextual para ver el HTML responsable de esta parte de la página. Esto te ayudará a analizar el HTML para tus programas de raspado web.

No utilices expresiones regulares para analizar HTML

Localizar un fragmento específico de HTML (o XML, JSON, TOML o YAML) en una cadena parece un caso perfecto para las expresiones regulares. Sin embargo, no te lo aconsejo. El HTML puede estar formateado de varias maneras y seguir siendo válido, pero intentar capturar todas las posibles variaciones en una expresión regular es tedioso y propenso a errores. Usar un módulo diseñado específicamente para analizar HTML, como bs4, tiene menos probabilidades de fallar.

Encontrarás una amplia argumentación de por qué no deberías analizar HTML con expresiones regulares en <https://>

Buscar elementos HTML

Una vez que el programa ha descargado una página web con el módulo requests, tenemos el contenido HTML de la página como un solo valor de cadena. Ahora tenemos que ver qué parte del HTML se corresponde con la información de la página web que nos interesa. Aquí es donde las herramientas para desarrolladores pueden ayudarnos. Supongamos que queremos escribir un programa para extraer información meteorológica de <https://weather.gov>. Antes de escribir código, hay que investigar un poco. Si visitamos el sitio y buscamos el código postal 94105, debería llevarnos a la página que muestra la previsión para esa zona.

¿Y si queremos extraer la información meteorológica de esa zona? Hacemos clic con el botón secundario (o **Control-clic** en macOS) en esa información y seleccionamos Inspeccionar en el menú contextual. Se abrirá la ventana de herramientas para desarrolladores, que muestra el HTML que produce esa parte en concreto de la página web. La figura 13.4 muestra las herramientas para desarrolladores abiertas en el HTML de la previsión más cercana. Ten en cuenta que, si <https://weather.gov> cambia el diseño del sitio, tendrás que repetir el proceso para inspeccionar los elementos nuevos.

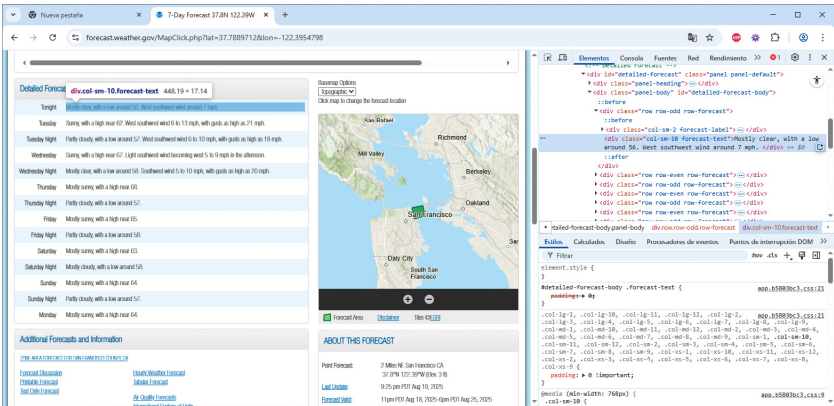


Figura 13.4. Inspección del elemento que contiene texto de la previsión.

Desde las herramientas para desarrolladores, podemos ver que el HTML responsable de la parte de predicción de la página es este:

```
<div class="col-sm-10 forecast-text"> Mostly clear, with a low  
around 56. West southwest wind around 7 mph. </div>
```

¡Esto es justo lo que estábamos buscando! Parece que la información está dentro de un elemento `<div>` con la clase CSS `forecast-text`.

Haz clic con el botón secundario en la consola de desarrollador del navegador y, en el menú que aparecerá, selecciona Copiar > Selector CSS. Esta opción copia una cadena como `'div.row-odd:nth-child(1) > div:nth-child(2)'` en el portapapeles. Podemos pasársela al método `select()` de BeautifulSoup o al método `find_element()` de Selenium, como veremos más adelante en este capítulo, para buscar el elemento en la cadena.

La sintaxis del selector CSS empleada en esta cadena especifica qué elementos HTML recuperar de una página web. La sintaxis completa del selector queda fuera del ámbito de este libro, pero puedes acceder al selector desde las herramientas para desarrolladores, como hemos hecho aquí. XPath es otra sintaxis para seleccionar elementos HTML, pero también está fuera del alcance del libro.

Ten en cuenta que, cuando un sitio web cambie su diseño, tendrás que actualizar las etiquetas HTML que comprueba tus *scripts*. Esto puede ocurrir sin aviso, así que echa un vistazo a tu programa por si de repente da errores por no poder encontrar elementos. En general, es mejor usar una API de un sitio web si la ofrece, ya que es mucho menos probable que cambie que el sitio en sí.

Analizar HTML con BeautifulSoup

Beautiful Soup es un paquete para extraer información de una página HTML. Para instalar el paquete usaremos el nombre `beautifulsoup4`, pero para importarlo emplearemos el nombre abreviado `bs4`. En este apartado, utilizaremos BeautifulSoup para analizar (en realidad, analizar y extraer las partes) el archivo HTML en <https://autbor.com/example3.html>, que tiene el siguiente contenido:

<!-- Esto es un comentario en HTML. -->

<html>

<head>

<title> Example Website Title</title>

<style>

.slogan {

color: gray;

font-size: 2em;

}

</style>

</head>

<body>

<h1>Example Website</h1>

<p>This <p>tag puts content into a
<i>single</i> paragraph.</p>

<p>This text is a
link to books by <span id =

"author">Al Sweigart.</p>

<p></p>

<p class="slogan">Learn to program in Python!</p>

<form>

<p><label>Username: <input id="login_user"
placeholder="admin" /></label></p>

```

<p> <label> Password: <input id="login_pass"
type="password" placeholder="swordfish" />

</form>

</label> </p>

<p> <label> Agree to disagree: <input type="checkbox" /> </
label> <input type="submit"
value="Fake Button" /> </p>

</body>

</html>

```

Ten en cuenta que el formulario de inicio de sesión de esta página es falso y se incluye solo para dar realismo.

Incluso un archivo HTML sencillo implica varias etiquetas y atributos diferentes, y la cosa se complica enseguida cuando se trata de sitios web complejos. Por suerte, Beautiful Soup hace que sea mucho más fácil trabajar con HTML.

Crear un objeto Beautiful Soup

La función `bs4.BeautifulSoup()` acepta una cadena que contiene el HTML que analizará y, después, devuelve un objeto `BeautifulSoup`. Por ejemplo, introduce lo siguiente en el intérprete de comandos interactivo mientras tu ordenador está conectado a Internet:

```

> > > import requests, bs4

> > > res = requests.get('https://autbor.com/example3.html')

> > > res.raise_for_status()

> > > example_soup = bs4.BeautifulSoup(res.text,
'html.parser')

> > > type(example_soup)

```

```
<class 'bs4.BeautifulSoup'>
```

Este código utiliza `requests.get()` para descargar la página principal del sitio web de este libro y pasa el atributo `text` de la respuesta a `bs4.BeautifulSoup()`. `Beautiful Soup` puede analizar distintos formatos y el argumento `'html.parser'` indica que estamos analizando HTML. Por último, el código guarda el objeto `BeautifulSoup` devuelto en una variable llamada `example_soup`.

También se puede cargar un archivo HTML desde el disco duro pasando un objeto `File` a `bs4.BeautifulSoup()`. Escribe lo siguiente en el intérprete de comandos interactivo (tras asegurarte de que el archivo `example3.html` está en el directorio de trabajo):

```
> > > import bs4
```

```
> > > with open('example3.html') as example_file:
```

```
... example_soup = bs4.BeautifulSoup(example_file,
'html.parser')
```

```
.
```

```
> > > type(example_soup)
```

```
<class 'bs4.BeautifulSoup'>
```

Cuando tenemos un objeto `BeautifulSoup`, podemos usar sus métodos para localizar partes específicas de un documento HTML.

Buscar un elemento

Podemos recuperar un elemento de una página web desde un objeto `BeautifulSoup` llamando a su método `select()` y pasándole una cadena de selector CSS para el elemento que estamos buscando. El método devuelve una lista de objetos `Tag`, que representa los elementos HTML coincidentes. La tabla 13.2 muestra ejemplos de los patrones de selector CSS más habituales con `select()`.

Tabla 13.2. Ejemplos de selectores CSS.

Selección. pasado al método `select()`

Todos los elementos `<div>`.

`soup.select('#author')` atributo `id` de `author`.

Todos los elementos que usan un atributo `class` de CSS llamado `notice`.

Todos los elementos llamados `<span>` que están dentro de un elemento llamado `<div>`.

Todos los elementos llamados `<span>` que están directamente dentro de un elemento llamado `<div>`, sin otros elementos entre medio.

Todos los elementos llamados `<input>` que tienen un atributo `name` con cualquier valor.

Todos los elementos llamados `<input>` que tienen un atributo `type` con el valor `button`.

Podemos combinar los distintos patrones de selector para hacer búsquedas sofisticadas. Por ejemplo, `soup.select('p #author')` busca cualquier elemento con un atributo `id` que sea `author`, siempre y cuando esté dentro de un elemento `<p>`.

Podemos pasar valores de etiqueta a la función `str()` para mostrar las etiquetas HTML que representan. Los valores de etiqueta también tienen un atributo `attrs` que contiene todos sus atributos HTML como un diccionario. Para ver un ejemplo, descarga la página <https://autbor.com/example3.html> como `example3.html` e introduce esto en el intérprete de comandos interactivo:

```
> > > import bs4
```

```
> > > example_file = open('example3.html')
```

```
> > > example_soup = bs4.BeautifulSoup(example_file.read(),  
'html.parser')
```

```
> > > elems = example_soup.select('#author')
```

```
> > > type(elems) # elems es una lista de objetos Tag.
```

```
<class 'bs4.element.ResultSet'>
```

```
> > > len(elems)
```



```
> > > type(elems[0])
```

```
<class 'bs4.element.Tag'>
```

```
> > > str(elems[0]) # El objeto Tag como cadena
```

```
'<span id="author">Al Sweigart</span>'
```

```
> > > elems[0].gettext() # El texto interno del elemento
```

```
'Al Sweigart'
```

```
> > > elems[0].attrs
```

```
{'id': 'author'}
```

Este código busca el elemento con `id="author"` en nuestro HTML de ejemplo. Usamos `select('#author')` para devolver una lista de todos los elementos con `id="author"`. A continuación, guardamos esta lista de objetos Tag en la variable `elems`. Ejecutar `len(elems)` nos dice que hay un objeto Tag en la lista, lo que significa que ha habido una coincidencia. Pasar el elemento a `str()` devuelve una cadena con las etiquetas de apertura y cierre y el texto del elemento. Llamar a `gettext()` con el elemento devuelve el texto del mismo, o el contenido entre las etiquetas de apertura y cierre: en este caso, `'Al Sweigart'`. Por último, `attrs` nos da un diccionario con el atributo del elemento, `'id'`, y el valor de este atributo, `'author'`.

También podemos sacar todos los elementos `<p>` del objeto BeautifulSoup. Introduce esto en el intérprete de comandos interactivo:

```
> > > p_elems = example_soup.select('p')
```

```
> > > str(p_elems[0])
```

```
'<p>This &lt;p&gt; tag puts <b>content</b> into a  
<i>single</i> paragraph.</p>'
```

```
> > > p_elems[0].gettext()
```

```
'This <p> tag puts content into a single paragraph.'
```

```
> > > str(p_elems[1])
```

```
'<p> <a href="https://inventwithpython.com/">This text is a  
link</a> to books by
```

```
<span id="author">Al Sweigart</span>.</p>'
```

```
> > > p_elems[1].gettext()
```

```
'This text is a link to books by Al Sweigart.'
```

```
> > > str(p_elems[2])
```

```
'<p>  </p>'
```

```
> > > p_elems[2].gettext()
```

```
“
```

Esta vez, `select()` nos da una lista de tres coincidencias, que almacenamos en `p_elems`. Utilizar `str()` con `p_elems[0]`, `p_elems[1]` y `p_elems[2]` muestra cada elemento como una cadena, y usar `gettext()` con cada elemento muestra su texto.

Obtener datos de atributos del elemento

El método `get()` para objetos `Tag` permite acceder a los valores de atributo HTML de un elemento. Pasaremos al método un nombre de atributo como cadena y recibiremos el valor de ese atributo. Usando `example3.html` de <https://author.com/example3.html>, escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > import bs4
```

```
> > > soup = bs4.BeautifulSoup(open('example3.html'),  
'html.parser')
```

```
> > > span_elem = soup.select('span')[0]
```

```
> > > str(span_elem)
```

```
'<span id="author">Al Sweigart</span>'
```

```
> > > span_elem.get('id')
```

```
'author'
```

```
> > > span_elem.get('some_nonexistent_attr') = None
```

```
True
```

```
> > > span_elem.attrs
```

```
{'id': 'author'}
```

Aquí, utilizamos `select()` para buscar cualquier elemento `<span>` y almacenamos el primer elemento encontrado en `span_elem`. Pasar el nombre de atributo `'id'` a `get()` devuelve el valor del atributo, `'author'`.

Proyecto 7: Abrir todos los resultados de una búsqueda

Cuando busco un tema en un motor de búsqueda, no miro solo un resultado cada vez. Haciendo clic con el botón central en el enlace de un resultado (o haciendo clic mientras mantengo pulsada la tecla **Control**), voy abriendo los primeros enlaces en un puñado de pestañas nuevas para leer después. Busco tantas cosas en Internet que este flujo (abrir el navegador, buscar el tema, ir haciendo clic en varios enlaces de uno en uno) es tedioso. Estaría bien si pudiera simplemente introducir un término en la línea de comandos y que el ordenador abriese automáticamente los primeros resultados en nuevas pestañas del navegador.

Vamos a escribir un *script* para que haga esto con la página de resultados de búsqueda de Python Package Index (<https://pypi.org>). Podrías adaptar un programa como este para otros sitios web, aunque Google, DuckDuckGo, Amazon y otros sitios web grandes suelen tomar medidas que hacen difícil el raspado de sus páginas de resultados. Esto es lo que debería hacer el programa:

- Obtener términos de búsqueda de los argumentos de línea de comandos.

- Recuperar la página de resultados de búsqueda.
 - Abrir una pestaña del navegador para cada resultado.
- Esto significa que el código tiene que hacer lo siguiente:
- Leer los argumentos de línea de comandos de sys.argv.
 - Buscar la página de resultados con el módulo requests.
 - Buscar los enlaces a cada resultado de la búsqueda.
 - Llamar a la función webbrowser.open() para abrir el navegador web.

Abre una pestaña nueva en el editor de archivos y guarda como searchpypi.py.

Paso 1: abrir la página de búsqueda

Antes de escribir código, hay que saber la URL de la página de resultados de búsqueda. Mirando la barra de dirección del navegador después de hacer una búsqueda, vemos una página con una URL parecida a esta: [https://pypi.org/search/?q= <TÉRMINO_DE_BÚSQUEDA>](https://pypi.org/search/?q=<TÉRMINO_DE_BÚSQUEDA>). El módulo requests puede descargar esta página web; después, podemos usar BeautifulSoup para buscar los enlaces de los resultados de la búsqueda en el HTML. Por último, utilizaremos el módulo webbrowser para abrir esos enlaces en pestañas del navegador.

Escribe el siguiente código:

```
# searchpypi.py – Abre varios resultados de una búsqueda en
pypi.org
```

```
import requests, sys, webbrowser, bs4
```

```
print("Searching...") # Muestra texto mientras descarga la página de
resultados de la búsqueda.
```

```
res = requests.get('https://pypi.org/search/?q=' + '
'.join(sys.argv[1:]))
```

```
res.raise_for_status()
```

```
# PENDIENTE: Recuperar los enlaces de los primeros resultados.
```

```
# PENDIENTE: Abrir una pestaña del navegador para cada resultado.
```

El usuario especificará los términos de búsqueda como argumentos de línea de comandos al abrir el programa y el código guardará esos argumentos como cadenas en una lista en `sys.argv`.

Paso 2: buscar todos los resultados

Ahora tenemos que usar Beautiful Soup para extraer los enlaces de los primeros resultados del HTML descargado. ¿Pero cómo sabemos cuál es el selector adecuado para el trabajo? Por ejemplo, no podemos simplemente buscar todas las etiquetas `<a>` porque hay un montón de enlaces que no nos interesan en el HTML. Habrá que inspeccionar la página de resultados de búsqueda con las herramientas para desarrolladores del navegador en busca de un selector que coja solo los enlaces que queremos. Después de buscar **pyautogui**, podemos abrir las herramientas para desarrolladores e inspeccionar los elementos de enlaces de la página. Pueden parecer complicados, como `<a class="package-snippet" href="/project/pyautogui">`, pero no importa que el elemento parezca increíblemente complicado: solo necesitamos ver el patrón que tienen todos los resultados de la búsqueda. Añade esto al código:

```
# searchpypi.py - Abre varios resultados de una búsqueda en pypi.org
```

```
import requests, sys, webbrowser, bs4
```

```
--fragmento omitido--
```

```
# Recupera los enlaces de los primeros resultados.
```

```
soup = bs4.BeautifulSoup(res.text, 'parser.html')
```

```
# Abre una pestaña del navegador para cada resultado.
```

```
link_elems = soup.select('.package-snippet')
```

Si miras los elementos `<a>`, verás que los enlaces de todos los resultados tienen `class="package-snippet"`. Mirando el resto del código fuente HTML, parece que la clase `package-snippet` se usa solo para los resultados de búsqueda. No hace falta saber qué es la clase CSS `package-snippet` ni para que sirve; solo la vamos a usar como marcador para el elemento `<a>` que estamos buscando.

Podemos crear un objeto BeautifulSoup a partir del texto HTML de la página descargada y usar el selector `'package-snippet'` para ver todos los elementos `<a>` que están dentro de un elemento que tiene la clase CSS `package-snippet`. Ten en cuenta que, si el sitio web PyPI cambia su diseño, es posible que tengas que actualizar este programa con una nueva cadena selector CSS que pasar a `soup.select()`. El resto del programa debería seguir actualizado.

Paso 3: abrir navegadores para cada resultado

Por último, debemos decir al programa que abra pestañas de navegador para los resultados. Añade esto al final del programa:

```
# searchpypi.py - Opens several search results on pypi.org  
  
import requests, sys, webbrowser, bs4  
  
--fragmento omitido--  
  
# Abre una pestaña del navegador para cada resultado.  
  
link_elems = soup.select('.package-snippet')  
  
num_open = min(5, len(link_elems))  
  
for i in range(num_open):  
  
url_to_open = 'https://pypi.org' + link_elems[i].get('href')  
  
print('Opening', url_to_open)  
  
webbrowser.open(url_to_open)
```

Por defecto, el programa abre los cinco primeros resultados en pestañas nuevas utilizando el módulo `webbrowser`. No obstante, el usuario puede haber buscado algo que genere menos de cinco resultados. La llamada a `soup.select()` devuelve una lista con todos los elementos que coincidían con el selector `'package-snippet'`, así que el número de pestañas que hay que abrir es o 5 o la longitud de la lista (el que sea más bajo).

La función integrada de Python `min()` devuelve el menor de los argumentos enteros o flotantes que se le pasan. (También hay una función integrada `max()` que devuelve el argumento más grande que se le pasa). Podemos usar `min()` para ver si hay menos de cinco enlaces en la lista y guardar el número de enlaces que abrir en una variable llamada `num_open`. A continuación, podemos ejecutar un bucle `for` llamando a `range(num_open)`.

En cada iteración del bucle, el código utiliza `webbrowser.open()` para abrir una pestaña nueva en el navegador web. Observa que el valor del atributo `href` en los elementos `<a>` devueltos no tiene la parte inicial <https://pypi.org>, así que hay que concatenarla con el valor de cadena del atributo `href`.

Ya podemos abrir al instante los primeros cinco resultados de la búsqueda en PyPI de, por ejemplo “boring stuff” ejecutando `searchpypi boring stuff` en la línea de comandos. Consulta el capítulo 12 para ver cómo ejecutar programas fácilmente en tu sistema operativo.

Ideas para programas similares

La ventaja de navegar con pestañas es que puedes abrir fácilmente varios enlaces a la vez para verlos luego. Un programa que haga esto puede ser un buen atajo para:

- Abrir todas las páginas de producto después de buscar en un comercio electrónico como Amazon.
- Abrir todos los enlaces a reseñas para un mismo producto.
- Abrir los enlaces de resultados de fotos después de buscar en un sitio como Flickr o Imgur.

Proyecto 8: Descargar cómics de XKCD

Los blogs, las páginas web de cómics y otros sitios que se actualizan con frecuencia suelen tener una página principal con el contenido publicado recientemente, además de un botón que conduce a la publicación anterior, que tendrá a su vez otro botón para ir a la anterior y así sucesivamente, creando un camino desde el post más nuevo hasta el más antiguo del sitio. Si quisiéramos una copia del contenido del sitio para leer sin conexión, podríamos navegar manualmente por cada página y guardarlas de una en una. Pero es un trabajo bastante aburrido, así que vamos a escribir un programa que lo haga por nosotros.

XKCD, que se muestra en la figura 13.5, es un cómic web popular entre los *geeks* que encaja en esta estructura. La página principal, <https://xkcd.com>, tiene un botón Prev que lleva al usuario a números anteriores. Descargar cada cómic a mano nos llevaría una eternidad, pero podemos escribir un *script* que lo haga en un par de minutos.

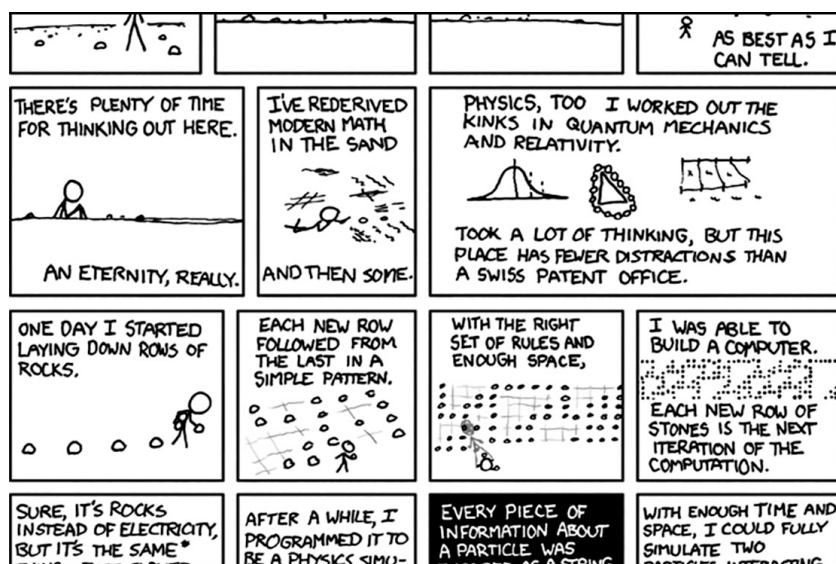


Figura 13.5. XKCD, “un cómic web de romance, sarcasmo, matemáticas y lenguaje”.

Esto es lo que debería hacer el programa:

- Cargar la página de inicio de XKCD.
- Guardar la imagen del cómic de esa página.
- Seguir el enlace al cómic anterior.
- Repetir hasta llegar al primer cómic o alcanzar el límite de descargas.

Esto significa que el código tendrá que hacer lo siguiente:

- Descargar páginas con el módulo requests.
- Buscar la URL de la imagen del cómic de cada página con BeautifulSoup.
- Descargar y guardar la imagen del cómic en el disco duro con `iter_content()`.
- Buscar la URL del enlace al cómic anterior y repetir.

Abre el editor de archivos y guarda el programa como `downloadXkcdComics.py`.

Paso 1: diseñar el programa

Si abrimos las herramientas para desarrolladores del navegador e inspeccionamos los elementos de la página, deberíamos ver lo siguiente:

- El atributo de un elemento `<img>` guarda la URL del archivo de imagen del cómic.
- El elemento `<img>` está dentro de un elemento `<div id="comic">`.
- El botón Prev tiene un atributo HTML `rel` con el valor `prev`.
- El botón Prev del cómic más antiguo enlaza con la URL <https://xkcd.com/#>, lo que indica que ya no hay más páginas anteriores.

Para evitar que los lectores de este libro se coman demasiado ancho

de banda del sitio web de XKCD, vamos a limitar el número de descargas que hacemos a 10 por defecto. Escribe el siguiente código:

```
# downloadXkcdComics.py – Descarga cómics de XKCD

import requests, os, bs4, time

url = 'https://xkcd.com' # URL de inicio

os.makedirs('xkcd', exist_ok=True) # Guarda cómics en ./xkcd

num_downloads = 0

MAX_DOWNLOADS = 10

while not url.endswith('#') and num_downloads <
MAX_DOWNLOADS:

# PENDIENTE: Descargar la página.

# PENDIENTE: Buscar la URL de la imagen del cómic.

# PENDIENTE: Descargar la imagen.

# PENDIENTE: Guardar la imagen en ./xkcd.

# PENDIENTE: Obtener la URL del botón Prev.

print('Done.')
```

El programa crea una variable `url` que empieza con el valor `'https://xkcd.com'` y lo actualiza repetidamente (en un bucle `while`) con la URL del botón Prev de la página actual. En cada paso por el bucle, descargamos el cómic de `url`. El bucle se detiene cuando `url` termina con `'#'` o hemos llegado a `MAX_DOWNLOADS`.

Descargaremos los archivos de imagen en una carpeta en el directorio de trabajo actual llamada `xkcd`. La llamada a `os.makedirs()` se asegura de que esta carpeta existe y el argumento de palabra clave `exist_ok=True` evita que la función lance una excepción si esta carpeta ya está creada.

Paso 2: descargar la página web

Vamos a implementar el código para descargar la página. Añade esto al programa:

```
# downloadXkcdComics.py - Descarga cómics de XKCD

import requests, os, bs4, time

url = 'https://xkcd.com' # URL de inicio

os.makedirs('xkcd', exist_ok=True) # Guarda cómics en ./xkcd

num_downloads = 0

MAX_DOWNLOADS = 10

while not url.endswith('#') and num_downloads <
MAX_DOWNLOADS:

    # Descarga la página.

    print(f'Downloading page {url}...')

    res = requests.get(url)

    res.raise_for_status()

    soup = bs4.BeautifulSoup(res.text, 'html.parser')

    # PENDIENTE: Buscar la URL de la imagen del cómic.

    # PENDIENTE: Descargar la imagen.

    # PENDIENTE: Guardar la imagen en ./xkcd.

    # PENDIENTE: Obtener la URL del botón Prev.

    print('Done.')
```

Primero, imprimimos url para que el usuario sepa qué URL está a punto de descargar el programa; después, usamos la función requests.get() del módulo requests para descargarla. Como siempre,

habría que llamar inmediatamente al método `raise_for_status()` del objeto `Response` para lanzar una excepción y terminar el programa si algo sale mal con la descarga. De otro modo, crea un objeto `BeautifulSoup` a partir del texto de la página descargada.

Paso 3: buscar y descargar la imagen del cómic

Para descargar el cómic de cada página, añade lo siguiente al programa:

```
# downloadXkcdComics.py - Descarga cómics de XKCD
```

```
import requests, os, bs4, time
```

```
--fragmento omitido--
```

```
# Busca la URL de la imagen del cómic.
```

```
comic_elem = soup.select('#comic img')
```

```
if comic_elem == :
```

```
    print('Could not find comic image.')
```

```
else:
```

```
    comic_URL = 'https:' + comic_elem[0].get('src')
```

```
    # Descargar la imagen.
```

```
    print(f'Downloading image {comic_URL}...')
```

```
    res = requests.get(comic_URL)
```

```
    res.raise_for_status()
```

```
    # PENDIENTE: Guardar la imagen en ./xkcd.
```

```
    # PENDIENTE: Obtener la URL del botón Prev.
```

```
    print('Done.')
```

Como hemos inspeccionado la página de inicio de XKCD con las

herramientas para desarrolladores, sabemos que el elemento `<img>` para la imagen del cómic está dentro de otro elemento con el atributo `id` establecido en `comic`, así que el selector `'#comic img'` nos dará el elemento `<img>` correcto a partir del objeto BeautifulSoup.

Hay unas cuantas páginas de XKCD con contenido especial que no es solo un archivo de imagen. No pasa nada; nos las saltaremos. Si el selector no encuentra elementos, `soup.select('#comic img')` devolverá un objeto `ResultSet` con una lista vacía. Cuando sucede eso, el programa puede imprimir un mensaje de error y avanzar sin descargar la imagen.

De lo contrario, el selector devolverá una lista con un elemento `<img>`. Podemos obtener el atributo `src` de este elemento `<img>` y pasarlo a `requests.get()` para descargar el archivo de imagen del cómic.

Paso 4: guardar la imagen y buscar el cómic anterior

En este momento, el archivo de imagen del cómic está guardado en la variable `res`. Tenemos que escribir estos datos de imagen en un archivo del disco duro. Añade este código al programa:

```
# downloadXkcdComics.py - Descarga cómics de XKCD
```

```
import requests, os, bs4, time
```

```
--fragmento omitido--
```

```
# Guarda la imagen en ./xkcd.
```

```
image_file = open(os.path.join('xkcd',  
os.path.basename(comic_URL)), 'wb')
```

```
for chunk in res.iter_content(100000):
```

```
image_file.write(chunk)
```

```
image_file.close()
```

```
# Obtiene la URL del botón Prev.
```

```
prev_link = soup.select('a[rel="prev"]')[0]
url = 'https://xkcd.com' + prev_link.get('href')
num_downloads = 1
time.sleep(1) # Pause so we don't hammer the web server.
print('Done.')
```

También necesitaremos un nombre de archivo para el archivo de imagen local que pasaremos a `open()`. `comic_URL` tendrá un valor como `'https://imgs.xkcd.com/comics/heartbleed_explanation.png'`, que como verás se parece mucho a una ruta de archivo. De hecho, podemos llamar a `os.path.basename()` con `comic_URL` para que devuelva solo la última parte de la URL, `'heartbleed_explanation.png'`, y use eso como nombre de archivo al guardar la imagen en el disco duro. Uniremos este nombre con el de la carpeta `xkcd` usando `os.path.join()` para que el programa utilice barras invertidas (`\`) en Windows y oblicuas (`/`) en macOS y Linux. Ahora que por fin tenemos el nombre de archivo, podemos llamar a `open()` para abrir un nuevo archivo en modo `'wb'`.

Recuerda que, como hemos visto antes en este capítulo, para guardar archivos descargados con `requests`, tenemos que pasar en bucle por el valor de retorno del método `iter_content()`. El código del bucle `for` escribe trozos de los datos de imagen en el archivo. A continuación, el código cierra el archivo, guardando la imagen en el disco duro. Después, el selector `'a[rel="prev"]'` identifica el elemento `<a>` con el atributo `rel` establecido en `prev`. Podemos usar el atributo `href` de este elemento `<a>` para obtener la URL del cómic anterior, que se guarda en `url`.

La última parte del código del bucle incrementa `num_downloads` en 1 para que no se descarguen todos los cómics por defecto. También introduce una pausa de un segundo con `time.sleep(1)` para evitar que el *script* “reviente” el sitio (es decir, descargue cómics a toda pastilla, lo que puede provocar problemas de rendimiento a otros visitantes del sitio web). Después, el bucle `while` empieza de nuevo todo el proceso de descarga. La salida de este programa tendría este aspecto:

Downloading page <https://xkcd.com...>

Downloading image https://imgs.xkcd.com/comics/phone_alarm.png...

Downloading page <https://xkcd.com/1358/...>

Downloading image <https://imgs.xkcd.com/comics/nro.png>...

Downloading page <https://xkcd.com/1357/...>

Downloading image https://imgs.xkcd.com/comics/free_speech.png...

Downloading page <https://xkcd.com/1356/...>

Downloading image https://imgs.xkcd.com/comics/orbital_mechanics.png...

Downloading page <https://xkcd.com/1355/...>

Downloading image https://imgs.xkcd.com/comics/airplane_message.png...

Downloading page <https://xkcd.com/1354/...>

Downloading image https://imgs.xkcd.com/comics/heartbleed_explanation.png...

--fragmento omitido--

Este proyecto es un buen ejemplo de programa que puede seguir enlaces automáticamente para extraer grandes cantidades de datos de Internet. Puedes aprender más de otras funcionalidades de Beautiful Soup con su documentación, en <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>.

Ideas para programas similares

Muchos programas de rastreo web implican descargar páginas y seguir enlaces. Otros programas similares a este podrían hacer lo siguiente:

- Hacer una copia de seguridad de un sitio web entero siguiendo todos sus enlaces.
- Copiar todos los mensajes de un foro web.
- Duplicar el catálogo de artículos en venta de un comercio en línea.

Los módulos `requests` y `bs4` son geniales siempre que podamos conseguir la URL que tenemos que pasar a `requests.get()`. Sin embargo, esta URL no siempre es fácil de encontrar. O tal vez el sitio por el que tenga que navegar el programa requiere iniciar sesión primero. Selenium dará a nuestros programas el poder de hacer tareas así de sofisticadas.

Controlar el navegador con Selenium

Selenium permite a Python controlar directamente el navegador haciendo clic programáticamente en enlaces y rellenando formularios, igual que haría un usuario humano. Con Selenium, podemos interactuar con páginas web de una forma mucho más avanzada que con `requests` y `Beautiful Soup`, pero, como abre un navegador web, es un poco más lento y difícil de ejecutar en segundo plano si, por ejemplo, necesitamos descargar archivos de la red.

Aun así, si necesitamos interactuar con una página web de una forma que, por ejemplo, dependa de código JavaScript que actualice la página, tendremos que usar Selenium en vez de `requests`. Esto se debe a que los principales sitios web de comercio electrónico, como Amazon, cuentan casi con toda seguridad con sistemas de software para reconocer el tráfico que sospechan que es un *script* que recopila su información o que se registra en varias cuentas gratuitas. Estos sitios pueden negarse a mostrarte las páginas después de un tiempo, rompiendo cualquier *script* que hayas creado. Selenium tiene muchas más posibilidades que `requests` de funcionar en estos sitios a largo plazo.

Una señal importante que indica a los sitios web que estás utilizando un *script* es la cadena del agente de usuario, que identifica el navegador web y se incluye en todas las solicitudes HTTP. Por ejemplo, la cadena del agente de usuario para el módulo

requests es algo como ‘python-requests/X.XX.X’. Puedes visitar un sitio como <https://www.whatsmyua.info> para ver tu cadena del agente de usuario. Con Selenium, es mucho más probable que pases por humano porque no solo es su cadena del agente de usuario la misma que la de un navegador normal (por ejemplo, ‘Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:108.0) Gecko/20100101 Firefox/108.0’), sino que además tiene los mismos patrones de tráfico: un navegador controlado por Selenium descargará imágenes, anuncios, *cookies* y rastreadores invasores de la intimidad igual que un navegador normal. No obstante, los sitios web pueden buscar la manera de detectar a Selenium, y los principales sitios web de venta de billetes y comercio electrónico suelen bloquearlo para impedir el rastreo de sus páginas.

Iniciar un navegador controlado por Selenium

Los siguientes ejemplos te mostrarán cómo controlar el navegador web Firefox. Si no tienes ya Firefox, puedes descargarlo de manera gratuita en <https://getfirefox.com>.

Importar módulos de Selenium es un poco difícil. En vez de import selenium, hay que ejecutar from selenium import webdriver. (El motivo exacto por el que Selenium funciona así queda fuera del ámbito de este libro). Después de eso, ya puedes abrir Firefox con Selenium. Introduce esto en el intérprete de comandos interactivo:

```
> > > from selenium import webdriver
```

```
> > > browser = webdriver.Firefox()
```

```
> > > type(browser)
```

```
<class 'selenium.webdriver.firefox.webdriver.WebDriver'>
```

```
> > > browser.get('https://inventwithpython.com')
```

Verás que, cuando se llama a webdriver.Firefox(), se abre el navegador Firefox. Llamar a type() con el valor webdriver.Firefox() revela que es del tipo de datos WebDriver. Y llamar a browser.get('https://inventwithpython.com') lleva al navegador a <https://inventwithpython.com>. Tu navegador debería tener un

aspecto parecido a la figura 13.6.

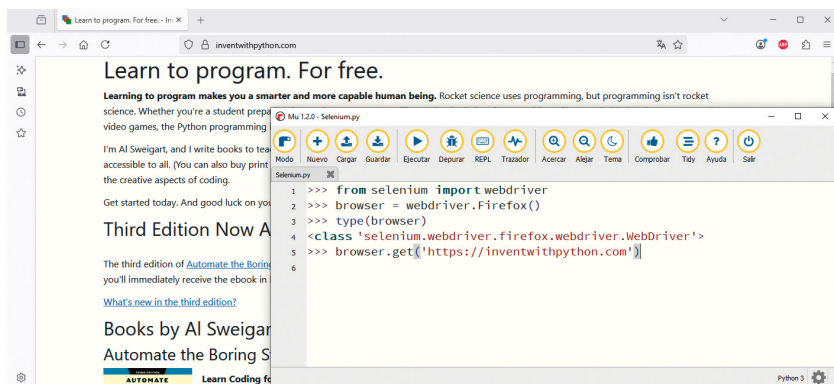


Figura 13.6. Cuando llamamos a `webdriver.Firefox()` y `get()` en Mu, aparece el navegador Firefox.

Si te sale el mensaje de error `geckodriver executable needs to be in PATH`, tendrás que descargar manualmente el controlador web de Firefox antes de poder usar Selenium para controlarlo. También puedes controlar otros navegadores si instalas su controlador y, en lugar de descargar manualmente los controladores web de los navegadores, puedes usar el paquete `webdriver-manager` de <https://pypi.org/project/webdriver-manager/>.

Hacer clic en los botones del navegador

Selenium puede simular clics en varios botones del navegador con estos métodos:

- `browser.back()`: Hace clic en el botón Anterior.
- `browser.forward()`: Hace clic en el botón Siguiente.
- `browser.refresh()`: Hace clic en el botón Recargar.
- `Browser.quit()`: Hace clic en el botón Cerrar.

Buscar elementos en la página

Un objeto `WebDriver` cuenta con los métodos `find_element()` y

`find_elements()` para buscar elementos en una página web. El método `find_element()` devuelve un solo objeto `WebElement`, que representa el primer elemento de la página que coincida con la búsqueda. El método `find_elements()` devuelve una lista de objetos `WebElement` para cada elemento coincidente de la página.

Podemos buscar elementos por su nombre de clase, selector CSS, u otros medios. Primero, ejecuta `from selenium.webdriver.common.by import By` para obtener el objeto `By`, que tiene varias constantes que podemos pasar a los métodos `find_element()` y `find_elements()`. La tabla 13.3 recoge estas constantes.

Tabla 13.3. Constantes `By` de Selenium para buscar elementos.

Objeto `WebElement` devuelto

`By.CLASS_NAME` son el *nombre* de clase CSS.

`By.CSS_SELECTOR` coinciden con el *selector* CSS.

`By.ID` elementos con un valor de atributo *id* coincidente.

`By.LINK_TEXT` > que coinciden completamente con el texto proporcionado.

`By.PARTIAL_LINK_TEXT` contienen el texto proporcionado.

`By.NAME` con un valor de atributo *name* coincidente.

`By.TAG_NAME` una etiqueta *name* coincidente (sin distinguir mayúsculas y minúsculas; un elemento <a> se encuentra con 'a' y con 'A').

Si no existen elementos en la página que coincidan con lo que está buscando el método, Selenium lanza una excepción `NoSuchElement`. Si no quieres que esta excepción detenga tu programa, añade declaraciones `try` y `except` a tu código.

Cuando ya tenemos el objeto `WebElement`, podemos saber más sobre él leyendo los atributos o llamando a los métodos de la tabla 13.4.

Tabla 13.4. Atributos y métodos `WebElement`.

Descripción método

`El nombre` de la etiqueta, como 'a' para un elemento <a>.

`El valor del atributo` *name* del elemento, como href en un elemento

<a>.

Get property (propiedad) Propiedad del elemento, que no aparece en el código HTML. Algunos ejemplos de propiedades HTML son innerHTML e innerText.

Text Texto dentro del elemento, como 'hello' en hello .

Clear Elementos de campo o área de texto, borra el texto introducido.

Displayed Devuelve True si el elemento es visible; si no, devuelve False.

Enabled Elementos de entrada, devuelve True si el elemento está habilitado; si no, devuelve False.

Selected Elementos de casillas de verificación o botones de opción, devuelve True si el elemento está seleccionado; si no, devuelve False.

Location Diccionario con claves 'x' e 'y' para la posición del elemento en la página.

Size Diccionario con claves 'width' y 'height' para el tamaño del elemento en la página.

Para ver un ejemplo, abre una pestaña nueva del editor de archivos y escribe este programa:

```
from selenium import webdriver

from selenium.webdriver.common.by import By

browser = webdriver.Firefox()

browser.get('https://autbor.com/example3.html')

elems = browser.find_elements(By.CSS_SELECTOR, 'p')

print(elems[0].text)

print(elems[0].get_property('innerHTML'))
```

Aquí, abrimos Firefox y lo llevamos a una URL. En esa página, obtenemos una lista de los elementos <p>, miramos el primero en index 0 y obtenemos la cadena de texto dentro de ese elemento <p>. A continuación, obtenemos la cadena de su propiedad innerHTML. La salida del programa es así:

This `<p>` tag puts content into a single paragraph.

This `<p>` tag puts `<b>` content `</b>` into a `<i>` single `</i>` paragraph.

El atributo `text` del elemento muestra el texto como lo veríamos en el navegador web: “This `<p>` tag puts content into a single paragraph”. También podemos examinar la propiedad `innerHTML` del elemento llamando al método `get_property()`, que es el código fuente HTML que incluye etiquetas y entidades HTML. (`<` y `>` son caracteres de escape HTML que representan los caracteres menor que `[ < ]` y mayor que `[ > ]`).

Observa que el atributo `text` es solo un atajo para llamar a `get_property('inner Text')`. Los nombres `innerHTML` e `innerText` son nombres de propiedades estándar para elementos HTML. En resumen, el código JavaScript y los controladores web acceden a estas propiedades, mientras que los atributos son parte del código fuente HTML, como `href` en `<a href = "https://inventwithpython.com">`.

Hacer clic en elementos de la página

Los objetos `WebElement` devueltos por los métodos `find_element()` y `find_elements()` tienen un método `click()` que simula un clic de ratón en ese elemento. Este método sirve para seguir un enlace, seleccionar un botón de opción, hacer clic en un botón Enviar o activar cualquier otra cosa que pase cuando se hace clic en el elemento. Para ver un ejemplo, introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > from selenium import webdriver
```

```
> > > from selenium.webdriver.common.by import By
```

```
> > > browser = webdriver.Firefox()
```

```
> > > browser.get('https://autbor.com/example3.html')
```

```
> > > link_elem = browser.find_element(By.LINK_TEXT, 'This  
text is a link')
```

```
> > > type(link_elem)
```

```
<class 'selenium.webdriver.remote.webelement.WebElement'>
```

```
> > > link_elem.click() # Sigue el enlace "This text is a link"
```

Este código abre Firefox en <https://autbor.com/example3.html>, obtiene el elemento WebElement para el elemento <a> con el texto "This is a link" y simula hacer clic en ese elemento <a> como si hubiésemos hecho clic nosotros; entonces el navegador abre el enlace.

Rellenar y enviar formularios

Enviar pulsaciones de teclas a los campos de texto de una página web es cuestión de encontrar el elemento <input> o <textarea> de ese campo y llamar al método keys(). Por ejemplo, escribe esto en el intérprete de comandos interactivo:

```
> > > from selenium import webdriver
```

```
> > > from selenium.webdriver.common.by import By
```

```
> > > browser = webdriver.Firefox()
```

```
> > > browser.get('https://autbor.com/example3.html')
```

```
> > > user_elem = browser.find_element(By.ID, 'login_user')
```

```
> > > user_elem.send_keys('your_real_username_here')
```

```
> > > password_elem = browser.find_element(By.ID, 'login_pass')
```

```
> > > password_elem.send_keys('your_real_password_here')
```

```
> > > password_elem.submit()
```

Siempre y cuando la página de inicio de sesión no haya cambiado el id del nombre de usuario y los elementos <input> de la contraseña, el código anterior encontrará esos campos de texto con el texto proporcionado. (Siempre puedes usar el inspector del

navegador para verificar el id). Llamar al método submit() con cualquier elemento tendrá el mismo resultado que hacer clic en el botón Enviar del formulario en el que esté el elemento. (Podríamos haber llamado con la misma facilidad a user_elem.submit() y el código habría hecho lo mismo).

Advertencia *Evita poner tus contraseñas en código fuente siempre que sea posible. Es fácil filtrarlas por accidente a otros cuando se dejan sin encriptar en tu disco duro.*

Enviar teclas especiales

Selenium tiene un módulo selenium.webdriver.common.keys para representar teclas, que guarda en atributos. Como el nombre del módulo es tan largo, es mucho más fácil ejecutar from selenium.webdriver.common.keys import Keys al principio del programa; de ese modo, basta con escribir Keys en cualquier parte donde normalmente habría que escribir selenium.webdriver.common.keys.

Podemos pasar a send_keys() cualquiera de las siguientes constantes:

Keys.~~DOWN~~UP

Keys.~~RETURN~~

Keys.~~ENTER~~SPACE

Keys.~~ENTER~~BETE

Keys.~~ENTER~~DOWN

También se puede pasar al método una cadena, como 'hello' o '?'.

Por ejemplo, si el cursor no está actualmente en un campo de texto, pulsar las teclas **Inicio** o **Fin** llevará al navegador al principio o al final de la página respectivamente. Introduce lo siguiente en el intérprete de comandos y observa cómo las llamadas a send_keys() mueven la página:

```
> > > from selenium import webdriver
```

```
> > > from selenium.webdriver.common.by import By
```

```
> > > from selenium.webdriver.common.keys import Keys  
> > > browser = webdriver.Firefox()  
> > > browser.get('https://nostarch.com')  
> > > html_elem = browser.find_element(By.TAG_NAME,  
'html')  
> > > html_elem.send_keys(Keys.END) # Baja al final  
> > > html_elem.send_keys(Keys.HOME) # Sube al principio
```

La etiqueta `<html>` es la etiqueta base en los archivos HTML: todo el contenido del archivo va entre las etiquetas `<html>` y `</html>`. Llamar a `browser.find_element(By.TAG_NAME, 'html')` es una buena forma de enviar teclas a la página web general a través de la etiqueta `<html>` principal. Esto sería útil si, por ejemplo, se carga contenido nuevo una vez que hemos llegado al final de la página.

Selenium puede hacer mucho más que las funciones descritas aquí. Puede modificar las *cookies* del navegador, sacar capturas de pantalla de páginas web y ejecutar JavaScript personalizados. Para saber más sobre estas funciones, visita la documentación de Selenium en <https://selenium-python.readthedocs.io>. También puedes encontrar charlas en congresos de Python sobre Selenium en el sitio web <https://pyvideo.org>.

Controlar el navegador con Playwright

Playwright es una librería de control de navegador similar a Selenium, pero más nueva. Aunque todavía no tiene el público tan amplio que tiene Selenium, presenta varias características que merece la pena conocer. La mejor de estas nuevas funciones es la capacidad de funcionar en “modo sin cabeza”, lo que significa que podemos simular un navegador sin abrir realmente la ventana del navegador en nuestra pantalla. Esto lo hace especialmente útil para ejecutar pruebas automatizadas o hacer raspados web en segundo plano. La documentación completa de Playwright se encuentra en <https://playwright.dev/python/docs/intro>.

Además, es más fácil instalar controladores para navegadores individuales con Playwright que con Selenium: simplemente ejecuta `python -m playwright install` en Windows y `python3 -m playwright install` en macOS y Linux desde una ventana de terminal para instalar los controladores para Firefox, Chrome y Safari. Como por lo demás Playwright es bastante similar a Selenium, no explicaré el raspado web y la información de los selectores CSS en este apartado.

Abrir un navegador controlado por Playwright

Una vez que tengas Playwright instalado, puedes probarlo con el siguiente programa:

```
from playwright.sync_api import sync_playwright

with sync_playwright() as playwright:

    browser = playwright.firefox.launch()

    page = browser.new_page()

    page.goto('https://autbor.com/example3.html')

    print(page.title())

    browser.close()
```

Al ejecutar el programa, hace una pausa mientras carga el navegador Firefox y abre la página <https://autbor.com/example3.html> y, a continuación, imprime el título: “Example Website”. También podemos usar `playwright.chromium.launch()` o `playwright.webkit.launch()` para utilizar Chrome y Safari, respectivamente.

Playwright llama automáticamente a los métodos `start()` y `stop()` cuando la ejecución entra y sale del bloque de la declaración `with`. Playwright tiene un modo síncrono, donde sus funciones no devuelven nada hasta que se completa la operación. De ese modo, no decimos accidentalmente al navegador que busque un elemento antes de que la página haya terminado de cargarse. Las funciones

asíncronas de Playwright quedan fuera del alcance de este libro.

Es posible que hayas observado que no ha aparecido ninguna ventana de navegador porque, por defecto, Playwright se ejecuta en modo sin cabeza. Esto, sumado a cómo Playwright pone código dentro de una declaración `with`, puede dificultar la depuración. Para ejecutar Playwright paso a paso, introduce esto en el intérprete de comandos interactivo:

```
> > > from playwright.sync_api import sync_playwright

> > > playwright = sync_playwright().start()

> > > browser = playwright.firefox.launch(headless=False,
slow_mo=50)

> > > page = browser.new_page()

> > > page.goto('https://autbor.com/example3.html')

<Response url='https://autbor.com/example3.html'
request= <Request

url='https://autbor.com/example3.html' method='GET'> >

> > > browser.close()

> > > playwright.stop()
```

Los argumentos `headless=False` y `slow_mo=50` de `playwright.firefox.launch()` hacen que aparezca la ventana del navegador en la pantalla y añaden un retardo de 50 milisegundos en sus operaciones para que podamos ver qué está pasando. No te preocupes por añadir pausas para dejar tiempo para que se carguen las páginas web: a Playwright se le da mucho mejor que a Selenium no moverse a la siguiente operación antes de que haya terminado la anterior.

El objeto `Page` devuelto por el método `new_page()` `Browser` representa una nueva pestaña en una nueva ventana del navegador. Podemos tener varias ventanas abiertas al mismo tiempo cuando usamos Playwright.

Hacer clic en los botones del navegador

Playwright puede simular clics en los botones del navegador llamando a los siguientes métodos Page con el objeto Page devuelto por `browser.new_page()`:

- `page.go_back()`: Hace clic en el botón Anterior.
- `page.go_forward()`: Hace clic en el botón Siguiente.
- `page.reload()`: Hace clic en el botón Recargar.
- `page.close()`: Hace clic en el botón Cerrar.

Buscar elementos en la página

Playwright tiene métodos de objeto Page denominados coloquialmente “localizadores” que devuelven objetos Locator, que representan posibles elementos HTML en una página web. Digo posibles porque, mientras que Selenium da error de inmediato si no puede encontrar el elemento que pedimos, Playwright entiende que la página podría crear dinámicamente el elemento después. Esto es útil, pero tiene un efecto secundario un poco negativo: si el elemento especificado no existe, Playwright hace una pausa de 30 segundos mientras espera a que aparezca.

Esta pausa es una lata si simplemente has cometido una errata. Para comprobar al instante si un elemento existe y está visible en la página, llama al método `is_visible()` con el objeto Locator devuelto por el localizador. También puedes llamar a `page.query_selector('selector')`, donde `selector` es una cadena del selector CSS o XPath del elemento. El método `page.query_selector()` vuelve inmediatamente y, si devuelve `None`, el elemento no existe ahora mismo en la página. Un objeto Locator puede coincidir con uno o varios elementos HTML de la página web. La tabla 13.5 contiene los localizadores de Playwright.

Tabla 13.5. Localizadores de Playwright para buscar elementos.

Objeto Locator devuelto

`page.get_by_<pool>()` (función opcionalmente `etiqueta`).

`page.get_by_text(texto)` en *texto* como parte de su texto interior.
`page.get_by_label(etiqueta)` *etiqueta* como etiqueta.
`page.get_by_placeholder(texto)` *texto* con valores de atributo `placeholder` coincidentes con el *texto* suministrado.
`page.get_by_alt_text(texto)` valores de atributo `alt` coincidentes con el *texto* suministrado.
`page.locator(selector)` selector CSS o XPath.

El método `get_by_role()` utiliza los roles ARIA (*Accessible Rich Internet Applications*, aplicaciones de Internet enriquecidas y accesibles), un conjunto de estándares que permiten al software identificar contenido de páginas web y adaptarlo para usuarios con problemas de visión y otras discapacidades. Por ejemplo, el rol “encabezado” se aplica a las etiquetas de `<h1>` a `<h6>`, siendo el texto entre `<h1>` y `</h1>` lo que podemos identificar con el parámetro `name` del método `get_by_role()`. (Los roles ARIA son mucho más complejos que esto, pero es un tema que queda fuera del alcance de este libro).

Podemos usar el texto entre las etiquetas de apertura y cierre para localizar elementos. Llamar a `page.get_by_text('is a link')` localizaría el elemento `<a>` en `<a href = "https://inventwithpython.com">This text is a link</a>`. Una coincidencia parcial, sin distinguir mayúsculas y minúsculas, suele ser lo bastante buena para localizar el elemento.

El método `page.get_by_label()` localiza elementos con el texto entre las etiquetas `<label>` y `</label>`. Por ejemplo, `page.get_by_label('Agree')` localizaría el elemento de casilla de verificación `<input>` en `<label>Agree to disagree: <input type="checkbox" /></label>`.

Las etiquetas `<input>` y `<textarea>` pueden tener un atributo `placeholder` que muestre texto marcador de posición hasta que el usuario introduzca texto de verdad. Por ejemplo, `page.get_by_placeholder('admin')` localizaría el elemento `<input>` de `<input id="login_user" placeholder="admin" />`.

Las imágenes de las páginas web pueden tener texto alternativo en su atributo `alt` para describir el contenido de la imagen a los usuarios invidentes. Algunos navegadores muestran esta descripción

como texto emergente al pasar el ratón por encima de la imagen. La llamada a `page.get_by_alt_text('Zophie')` devolvería el elemento `<img>` de ``.

Si solo necesitas obtener un objeto Locator con un selector CSS, llama a `locator()` y pásale la cadena del selector. Esto es parecido a usar el método `find_elements()` de Selenium con la constante `By.CSS_SELECTOR`.

Tabla 13.6. Métodos de Locator.

	Descripción
<code>get_attribute(nombre)</code>	Atributo <i>nombre</i> del elemento, como <code>'https://nostarch.com'</code> para el atributo <code>href</code> en un elemento <code></code> .
<code>count()</code>	Devuelve un entero del número de elementos coincidentes en este objeto Locator.
<code>nth(índice)</code>	Devuelve un objeto Locator del elemento coincidente dado por el índice. Por ejemplo, <code>nth(3)</code> devuelve el cuarto elemento coincidente, ya que el índice 0 es la primera coincidencia.
<code>first()</code>	Objeto Locator del primer elemento coincidente. Es lo mismo que <code>nth(0)</code> .
<code>last()</code>	Objeto Locator del último elemento coincidente. Si hay, por ejemplo, cinco elementos, esto sería lo mismo que <code>nth(4)</code> .
<code>all()</code>	Devuelve una lista de objetos Locator para cada elemento coincidente individual.
<code>get_text()</code>	Texto dentro del elemento, como <code>'hello'</code> en <code>hello</code> .
<code>get_html()</code>	Código fuente HTML dentro del elemento, como <code>'hello'</code> en <code>hello</code> .
<code>click()</code>	Un clic en el elemento, lo cual es útil para elementos de enlace, casilla de verificación y botón.
<code>is_visible()</code>	True si el elemento está visible; si no, devuelve False.
<code>is_enabled()</code>	Para los elementos de entrada, devuelve True si el elemento está habilitado; si no, devuelve False.
<code>is_checked()</code>	Para las casillas de verificación y botones de opción, devuelve True si el elemento está seleccionado; si no, devuelve False.
<code>bounding_box()</code>	Diccionario con claves <code>'x'</code> e <code>'y'</code> para la posición de la esquina superior izquierda del elemento en la página, junto con las

claves 'width' y 'height' para el tamaño del elemento.

Dado que los objetos Locator pueden representar múltiples elementos, podemos obtener un objeto Locator para un elemento individual con el método nth(), pasando el índice basado en cero. Para ver un ejemplo, abre una pestaña nueva en el editor de archivos e introduce el siguiente programa:

```
from playwright.sync_api import sync_playwright

with sync_playwright() as playwright:

    browser = playwright.firefox.launch(headless=False,
    slow_mo=50)

    page = browser.new_page()

    page.goto('https://autbor.com/example3.html')

    elems = page.locator('p')

    print(elems.nth(0).inner_text())

    print(elems.nth(0).inner_html())
```

Al igual que con el ejemplo de Selenium, la salida de este programa es esta:

This <p> tag puts content into a single paragraph.

This <p> tag puts content into a <i>single</i> paragraph.

El código page.locator('p') devuelve un objeto Locator que coincide con todos los elementos <p> en la página web, y la llamada al método nth(0) devuelve un objeto Locator solo para el primer elemento <p>. El objeto Locator también dispone de un método count() para devolver el número de elementos coincidentes en el localizador (similar a la función len() para listas de Python). También hay atributos first y last que contienen un localizador que coincide con el primer o el último elemento. Si quieres una lista de objetos Locator para cada elemento coincidente, llama al método

all()).

Cuando ya tengas objetos Locator para elementos, puedes hacer clics de ratón o pulsaciones de teclas con ellos, como veremos en los siguientes apartados.

Hacer clic en elementos de la página

El objeto Page tiene métodos click(), check(), uncheck() y set_checked() para simular clics en elementos de enlace, botón y casilla de verificación. Puedes llamar a estos métodos y pasarles la cadena de un selector CSS o XPath del elemento, o usar las funciones para Locator de Playwright recogidas en la tabla 13.6. Introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > from playwright.sync_api import sync_playwright
> > > playwright = sync_playwright().start()
> > > browser = playwright.firefox.launch(headless=False,
slow_mo=50)
> > > page = browser.new_page()
> > > page.goto('https://autbor.com/example3.html')
<Response url = 'https://autbor.com/example3.html'
request = <Request
url = 'https://autbor.com/example3.html' method = 'GET'> >
> > > page.click('input [type = checkbox] ') # Selecciona la
casilla de verificación
> > > page.click('input [type = checkbox] ') # Deselecciona la
casilla de verificación
> > > page.click('a') # Hace clic en el enlace
> > > page.go_back()
> > > checkbox_elem = page.get_by_role('checkbox') # Llama
```

a un método Locator

```
> > > checkbox_elem.check() # Selecciona la casilla de verificación
```

```
> > > checkbox_elem.uncheck() # Deselecciona la casilla de verificación
```

```
> > > checkbox_elem.set_checked(True) # Selecciona la casilla de verificación
```

```
> > > checkbox_elem.set_checked(False) # Deselecciona la casilla de verificación
```

```
> > > page.get_by_text('is a link').click() # Usa un método Locator
```

```
> > > browser.close()
```

```
> > > playwright.stop()
```

Los métodos `check()` y `uncheck()` son más fiables que `click()` para las casillas de verificación. El método `click()` cambia la casilla al estado contrario, mientras que `check()` y `uncheck()` la dejan seleccionada o deseleccionada al margen del estado en que estuviera antes. Del mismo modo, el método `set_checked()` permite pasar `True` para seleccionar la casilla o `False` para deseleccionarla.

Rellenar y enviar formularios

Los objetos Locator tienen un método `fill()` que coge una cadena y rellena el elemento `<input>` o `<textarea>` con ese texto. Esto es útil para rellenar formularios en línea, como el formulario de inicio de sesión en nuestro ejemplo `example3.html`:

```
> > > from playwright.sync_api import sync_playwright
```

```
> > > playwright = sync_playwright().start()
```

```
> > > browser = playwright.firefox.launch(headless=False, slow_mo=50)
```



```
> > > page = browser.new_page()

> > > page.goto('https://autbor.com/example3.html')

<Response url = 'https://autbor.com/example3.html'
request = <Request

url = 'https://autbor.com/example3.html' method = 'GET' > >

> > > page.locator('#login_user').fill('tu_nombre_de_usuario)

> > > page.locator('#login_pass').fill('tu_contraseña)

> > > page.locator('input [type = submit] ').click()

> > > browser.close()

> > > playwright.stop()
```

También hay un método `clear()` que borra todo el texto que esté en ese momento en el elemento. A diferencia de Selenium, no hay método `submit()` en Playwright, así que habrá que llamar a `click()` con el objeto Locator que coincida con el elemento del botón Enviar.

Enviar teclas especiales

También podemos simular pulsaciones de teclado en la página web con el método `press()` de los objetos Locator. Por ejemplo, si el cursor no se encuentra en ese momento en un campo de texto, pulsar las teclas **Inicio** y **Fin** llevará el navegador al principio o al final de la página, respectivamente. Escribe lo siguiente en el intérprete de comandos interactivo y observa cómo mueven la página las llamadas a `press()`:

```
> > > from playwright.sync_api import sync_playwright

> > > playwright = sync_playwright().start()

> > > browser = playwright.firefox.launch(headless = False,
slow_mo = 50)
```

```
> > > page = browser.new_page()
```

```
> > > page.goto('https://autbor.com/example3.html')
```

```
<Response url = 'https://autbor.com/example3.html'  
request = <Request
```

```
url = 'https://autbor.com/example3.html' method = 'GET' > >
```

```
> > > page.locator('html').press('End') # Baja al final
```

```
> > > page.locator('html').press('Home') # Sube al principio
```

```
> > > browser.close()
```

```
> > > playwright.stop()
```

Las cadenas que pasamos a `press()` pueden incluir cadenas de un solo carácter (como 'a' o '?'); las claves de modificación 'Shift', 'Control', 'Alt' o 'Meta' (como en 'Control + A', para **Control-A**); y cualquiera de las siguientes:

```
'Backspace'  
'End'  
'Enter'  
'Escape'  
'F1'  
'F12'  
'F13'  
'F14'  
'F15'  
'F16'  
'F17'  
'F18'  
'F19'  
'F20'  
'F21'  
'F22'  
'F23'  
'F24'  
'F25'  
'F26'  
'F27'  
'F28'  
'F29'  
'F30'  
'F31'  
'F32'  
'F33'  
'F34'  
'F35'  
'F36'  
'F37'  
'F38'  
'F39'  
'F40'  
'F41'  
'F42'  
'F43'  
'F44'  
'F45'  
'F46'  
'F47'  
'F48'  
'F49'  
'F50'  
'F51'  
'F52'  
'F53'  
'F54'  
'F55'  
'F56'  
'F57'  
'F58'  
'F59'  
'F60'  
'F61'  
'F62'  
'F63'  
'F64'  
'F65'  
'F66'  
'F67'  
'F68'  
'F69'  
'F70'  
'F71'  
'F72'  
'F73'  
'F74'  
'F75'  
'F76'  
'F77'  
'F78'  
'F79'  
'F80'  
'F81'  
'F82'  
'F83'  
'F84'  
'F85'  
'F86'  
'F87'  
'F88'  
'F89'  
'F90'  
'F91'  
'F92'  
'F93'  
'F94'  
'F95'  
'F96'  
'F97'  
'F98'  
'F99'  
'F100'  
'F101'  
'F102'  
'F103'  
'F104'  
'F105'  
'F106'  
'F107'  
'F108'  
'F109'  
'F110'  
'F111'  
'F112'  
'F113'  
'F114'  
'F115'  
'F116'  
'F117'  
'F118'  
'F119'  
'F120'  
'F121'  
'F122'  
'F123'  
'F124'  
'F125'  
'F126'  
'F127'  
'F128'  
'F129'  
'F130'  
'F131'  
'F132'  
'F133'  
'F134'  
'F135'  
'F136'  
'F137'  
'F138'  
'F139'  
'F140'  
'F141'  
'F142'  
'F143'  
'F144'  
'F145'  
'F146'  
'F147'  
'F148'  
'F149'  
'F150'  
'F151'  
'F152'  
'F153'  
'F154'  
'F155'  
'F156'  
'F157'  
'F158'  
'F159'  
'F160'  
'F161'  
'F162'  
'F163'  
'F164'  
'F165'  
'F166'  
'F167'  
'F168'  
'F169'  
'F170'  
'F171'  
'F172'  
'F173'  
'F174'  
'F175'  
'F176'  
'F177'  
'F178'  
'F179'  
'F180'  
'F181'  
'F182'  
'F183'  
'F184'  
'F185'  
'F186'  
'F187'  
'F188'  
'F189'  
'F190'  
'F191'  
'F192'  
'F193'  
'F194'  
'F195'  
'F196'  
'F197'  
'F198'  
'F199'  
'F200'  
'F201'  
'F202'  
'F203'  
'F204'  
'F205'  
'F206'  
'F207'  
'F208'  
'F209'  
'F210'  
'F211'  
'F212'  
'F213'  
'F214'  
'F215'  
'F216'  
'F217'  
'F218'  
'F219'  
'F220'  
'F221'  
'F222'  
'F223'  
'F224'  
'F225'  
'F226'  
'F227'  
'F228'  
'F229'  
'F230'  
'F231'  
'F232'  
'F233'  
'F234'  
'F235'  
'F236'  
'F237'  
'F238'  
'F239'  
'F240'  
'F241'  
'F242'  
'F243'  
'F244'  
'F245'  
'F246'  
'F247'  
'F248'  
'F249'  
'F250'  
'F251'  
'F252'  
'F253'  
'F254'  
'F255'  
'F256'  
'F257'  
'F258'  
'F259'  
'F260'  
'F261'  
'F262'  
'F263'  
'F264'  
'F265'  
'F266'  
'F267'  
'F268'  
'F269'  
'F270'  
'F271'  
'F272'  
'F273'  
'F274'  
'F275'  
'F276'  
'F277'  
'F278'  
'F279'  
'F280'  
'F281'  
'F282'  
'F283'  
'F284'  
'F285'  
'F286'  
'F287'  
'F288'  
'F289'  
'F290'  
'F291'  
'F292'  
'F293'  
'F294'  
'F295'  
'F296'  
'F297'  
'F298'  
'F299'  
'F300'  
'F301'  
'F302'  
'F303'  
'F304'  
'F305'  
'F306'  
'F307'  
'F308'  
'F309'  
'F310'  
'F311'  
'F312'  
'F313'  
'F314'  
'F315'  
'F316'  
'F317'  
'F318'  
'F319'  
'F320'  
'F321'  
'F322'  
'F323'  
'F324'  
'F325'  
'F326'  
'F327'  
'F328'  
'F329'  
'F330'  
'F331'  
'F332'  
'F333'  
'F334'  
'F335'  
'F336'  
'F337'  
'F338'  
'F339'  
'F340'  
'F341'  
'F342'  
'F343'  
'F344'  
'F345'  
'F346'  
'F347'  
'F348'  
'F349'  
'F350'  
'F351'  
'F352'  
'F353'  
'F354'  
'F355'  
'F356'  
'F357'  
'F358'  
'F359'  
'F360'  
'F361'  
'F362'  
'F363'  
'F364'  
'F365'  
'F366'  
'F367'  
'F368'  
'F369'  
'F370'  
'F371'  
'F372'  
'F373'  
'F374'  
'F375'  
'F376'  
'F377'  
'F378'  
'F379'  
'F380'  
'F381'  
'F382'  
'F383'  
'F384'  
'F385'  
'F386'  
'F387'  
'F388'  
'F389'  
'F390'  
'F391'  
'F392'  
'F393'  
'F394'  
'F395'  
'F396'  
'F397'  
'F398'  
'F399'  
'F400'  
'F401'  
'F402'  
'F403'  
'F404'  
'F405'  
'F406'  
'F407'  
'F408'  
'F409'  
'F410'  
'F411'  
'F412'  
'F413'  
'F414'  
'F415'  
'F416'  
'F417'  
'F418'  
'F419'  
'F420'  
'F421'  
'F422'  
'F423'  
'F424'  
'F425'  
'F426'  
'F427'  
'F428'  
'F429'  
'F430'  
'F431'  
'F432'  
'F433'  
'F434'  
'F435'  
'F436'  
'F437'  
'F438'  
'F439'  
'F440'  
'F441'  
'F442'  
'F443'  
'F444'  
'F445'  
'F446'  
'F447'  
'F448'  
'F449'  
'F450'  
'F451'  
'F452'  
'F453'  
'F454'  
'F455'  
'F456'  
'F457'  
'F458'  
'F459'  
'F460'  
'F461'  
'F462'  
'F463'  
'F464'  
'F465'  
'F466'  
'F467'  
'F468'  
'F469'  
'F470'  
'F471'  
'F472'  
'F473'  
'F474'  
'F475'  
'F476'  
'F477'  
'F478'  
'F479'  
'F480'  
'F481'  
'F482'  
'F483'  
'F484'  
'F485'  
'F486'  
'F487'  
'F488'  
'F489'  
'F490'  
'F491'  
'F492'  
'F493'  
'F494'  
'F495'  
'F496'  
'F497'  
'F498'  
'F499'  
'F500'  
'F501'  
'F502'  
'F503'  
'F504'  
'F505'  
'F506'  
'F507'  
'F508'  
'F509'  
'F510'  
'F511'  
'F512'  
'F513'  
'F514'  
'F515'  
'F516'  
'F517'  
'F518'  
'F519'  
'F520'  
'F521'  
'F522'  
'F523'  
'F524'  
'F525'  
'F526'  
'F527'  
'F528'  
'F529'  
'F530'  
'F531'  
'F532'  
'F533'  
'F534'  
'F535'  
'F536'  
'F537'  
'F538'  
'F539'  
'F540'  
'F541'  
'F542'  
'F543'  
'F544'  
'F545'  
'F546'  
'F547'  
'F548'  
'F549'  
'F550'  
'F551'  
'F552'  
'F553'  
'F554'  
'F555'  
'F556'  
'F557'  
'F558'  
'F559'  
'F560'  
'F561'  
'F562'  
'F563'  
'F564'  
'F565'  
'F566'  
'F567'  
'F568'  
'F569'  
'F570'  
'F571'  
'F572'  
'F573'  
'F574'  
'F575'  
'F576'  
'F577'  
'F578'  
'F579'  
'F580'  
'F581'  
'F582'  
'F583'  
'F584'  
'F585'  
'F586'  
'F587'  
'F588'  
'F589'  
'F590'  
'F591'  
'F592'  
'F593'  
'F594'  
'F595'  
'F596'  
'F597'  
'F598'  
'F599'  
'F600'  
'F601'  
'F602'  
'F603'  
'F604'  
'F605'  
'F606'  
'F607'  
'F608'  
'F609'  
'F610'  
'F611'  
'F612'  
'F613'  
'F614'  
'F615'  
'F616'  
'F617'  
'F618'  
'F619'  
'F620'  
'F621'  
'F622'  
'F623'  
'F624'  
'F625'  
'F626'  
'F627'  
'F628'  
'F629'  
'F630'  
'F631'  
'F632'  
'F633'  
'F634'  
'F635'  
'F636'  
'F637'  
'F638'  
'F639'  
'F640'  
'F641'  
'F642'  
'F643'  
'F644'  
'F645'  
'F646'  
'F647'  
'F648'  
'F649'  
'F650'  
'F651'  
'F652'  
'F653'  
'F654'  
'F655'  
'F656'  
'F657'  
'F658'  
'F659'  
'F660'  
'F661'  
'F662'  
'F663'  
'F664'  
'F665'  
'F666'  
'F667'  
'F668'  
'F669'  
'F670'  
'F671'  
'F672'  
'F673'  
'F674'  
'F675'  
'F676'  
'F677'  
'F678'  
'F679'  
'F680'  
'F681'  
'F682'  
'F683'  
'F684'  
'F685'  
'F686'  
'F687'  
'F688'  
'F689'  
'F690'  
'F691'  
'F692'  
'F693'  
'F694'  
'F695'  
'F696'  
'F697'  
'F698'  
'F699'  
'F700'  
'F701'  
'F702'  
'F703'  
'F704'  
'F705'  
'F706'  
'F707'  
'F708'  
'F709'  
'F710'  
'F711'  
'F712'  
'F713'  
'F714'  
'F715'  
'F716'  
'F717'  
'F718'  
'F719'  
'F720'  
'F721'  
'F722'  
'F723'  
'F724'  
'F725'  
'F726'  
'F727'  
'F728'  
'F729'  
'F730'  
'F731'  
'F732'  
'F733'  
'F734'  
'F735'  
'F736'  
'F737'  
'F738'  
'F739'  
'F740'  
'F741'  
'F742'  
'F743'  
'F744'  
'F745'  
'F746'  
'F747'  
'F748'  
'F749'  
'F750'  
'F751'  
'F752'  
'F753'  
'F754'  
'F755'  
'F756'  
'F757'  
'F758'  
'F759'  
'F760'  
'F761'  
'F762'  
'F763'  
'F764'  
'F765'  
'F766'  
'F767'  
'F768'  
'F769'  
'F770'  
'F771'  
'F772'  
'F773'  
'F774'  
'F775'  
'F776'  
'F777'  
'F778'  
'F779'  
'F780'  
'F781'  
'F782'  
'F783'  
'F784'  
'F785'  
'F786'  
'F787'  
'F788'  
'F789'  
'F790'  
'F791'  
'F792'  
'F793'  
'F794'  
'F795'  
'F796'  
'F797'  
'F798'  
'F799'  
'F800'  
'F801'  
'F802'  
'F803'  
'F804'  
'F805'  
'F806'  
'F807'  
'F808'  
'F809'  
'F810'  
'F811'  
'F812'  
'F813'  
'F814'  
'F815'  
'F816'  
'F817'  
'F818'  
'F819'  
'F820'  
'F821'  
'F822'  
'F823'  
'F824'  
'F825'  
'F826'  
'F827'  
'F828'  
'F829'  
'F830'  
'F831'  
'F832'  
'F833'  
'F834'  
'F835'  
'F836'  
'F837'  
'F838'  
'F839'  
'F840'  
'F841'  
'F842'  
'F843'  
'F844'  
'F845'  
'F846'  
'F847'  
'F848'  
'F849'  
'F850'  
'F851'  
'F852'  
'F853'  
'F854'  
'F855'  
'F856'  
'F857'  
'F858'  
'F859'  
'F860'  
'F861'  
'F862'  
'F863'  
'F864'  
'F865'  
'F866'  
'F867'  
'F868'  
'F869'  
'F870'  
'F871'  
'F872'  
'F873'  
'F874'  
'F875'  
'F876'  
'F877'  
'F878'  
'F879'  
'F880'  
'F881'  
'F882'  
'F883'  
'F884'  
'F885'  
'F886'  
'F887'  
'F888'  
'F889'  
'F890'  
'F891'  
'F892'  
'F893'  
'F894'  
'F895'  
'F896'  
'F897'  
'F898'  
'F899'  
'F900'  
'F901'  
'F902'  
'F903'  
'F904'  
'F905'  
'F906'  
'F907'  
'F908'  
'F909'  
'F910'  
'F911'  
'F912'  
'F913'  
'F914'  
'F915'  
'F916'  
'F917'  
'F918'  
'F919'  
'F920'  
'F921'  
'F922'  
'F923'  
'F924'  
'F925'  
'F926'  
'F927'  
'F928'  
'F929'  
'F930'  
'F931'  
'F932'  
'F933'  
'F934'  
'F935'  
'F936'  
'F937'  
'F938'  
'F939'  
'F940'  
'F941'  
'F942'  
'F943'  
'F944'  
'F945'  
'F946'  
'F947'  
'F948'  
'F949'  
'F950'  
'F951'  
'F952'  
'F953'  
'F954'  
'F955'  
'F956'  
'F957'  
'F958'  
'F959'  
'F960'  
'F961'  
'F962'  
'F963'  
'F964'  
'F965'  
'F966'  
'F967'  
'F968'  
'F969'  
'F970'  
'F971'  
'F972'  
'F973'  
'F974'  
'F975'  
'F976'  
'F977'  
'F978'  
'F979'  
'F980'  
'F981'  
'F982'  
'F983'  
'F984'  
'F985'  
'F986'  
'F987'  
'F988'  
'F989'  
'F990'  
'F991'  
'F992'  
'F993'  
'F994'  
'F995'  
'F996'  
'F997'  
'F998'  
'F999'  
'F1000'
```

Playwright tiene muchas más funciones que las descritas aquí. Para descubrirlas, visita la documentación de Playwright en <https://playwright.dev>. También puedes encontrar charlas en conferencias de Python sobre Playwright buscando en <https://pyvideo.org>.

Resumen

Las tareas más aburridas no se limitan a los archivos que tenemos en nuestro ordenador. Poder descargar páginas web con programación extiende nuestros programas a Internet. El módulo `requests` hace que sea fácil la descarga y, con unos conocimientos

básicos de conceptos y selectores HTML, podemos usar el módulo BeautifulSoup para analizar las páginas que queremos descargar.

Pero, para automatizar por completo cualquier tarea basada en la web, necesitas control directo de tu navegador a través de los paquetes Selenium y Playwright. Estos paquetes te permitirán iniciar sesión en sitios web y rellenar formularios automáticamente. Como un navegador web es la forma más habitual de enviar y recibir información por Internet, es una buena utilidad para tu caja de herramientas de programador.

Preguntas prácticas

1. Describe brevemente las diferencias entre los módulos webbrowser, requests y bs4.
2. ¿Qué tipo de objeto devuelve requests.get()? ¿Cómo podemos acceder al contenido descargado como un valor de cadena?
3. ¿Qué método de requests comprueba si la descarga ha ido bien?
4. ¿Cómo podemos obtener el código de estado HTTP de una respuesta de requests?
5. ¿Cómo guardamos una respuesta de requests en un archivo?
6. ¿En qué dos formatos devuelven principalmente sus respuestas las API en línea?
7. ¿Cuál es el atajo de teclado para abrir las herramientas para desarrolladores de un navegador?
8. ¿Cómo podemos ver (en las herramientas para desarrolladores) el HTML de un elemento específico de una página web?
9. ¿Qué cadena de selector CSS encontrará el elemento con un valor main para el atributo id?
10. ¿Qué cadena de selector CSS encontrará los elementos con un atributo id de highlight?
11. Supongamos que tienes un objeto Tag de BeautifulSoup

almacenado en la variable `spam` para el elemento `<div>Hello, world!</div>`. ¿Cómo puedes obtener una cadena 'Hello, world!' del objeto `Tag`?

12. ¿Cómo guardarías todos los atributos de un objeto `Tag` de `Beautiful Soup` en una variable llamada `link_elem`?

13. Ejecutar `import selenium` no funciona. ¿Cómo se importa `Selenium`?

14. ¿Cuál es la diferencia entre los métodos `find_element()` y `find_elements()` de `Selenium`?

15. ¿Con qué métodos cuentan los objetos `WebElement` de `Selenium` para simular clics de ratón y pulsaciones de teclas?

16. En `Playwright`, ¿qué localizador puede simular pulsar **Control-A** para seleccionar todo el texto de la página?

17. ¿Cómo puedes simular hacer clic en los botones Anterior, Siguiente y Recargar del navegador con `Selenium`?

18. ¿Cómo puedes simular hacer clic en los botones Anterior, Siguiente y Recargar del navegador con `Playwright`?

Programas de práctica

Para practicar, escribe programas que hagan las siguientes tareas.

Descargar imágenes de un sitio

Escribe un programa que abra un sitio web para compartir fotos, como Flickr o Imgur, busque una categoría y descargue todas las imágenes que encuentre ahí. Podrías escribir un programa que funcione con cualquier sitio de fotos que tenga función de búsqueda.

2048

El juego *2048* es un sencillo juego en el que combinas cajas

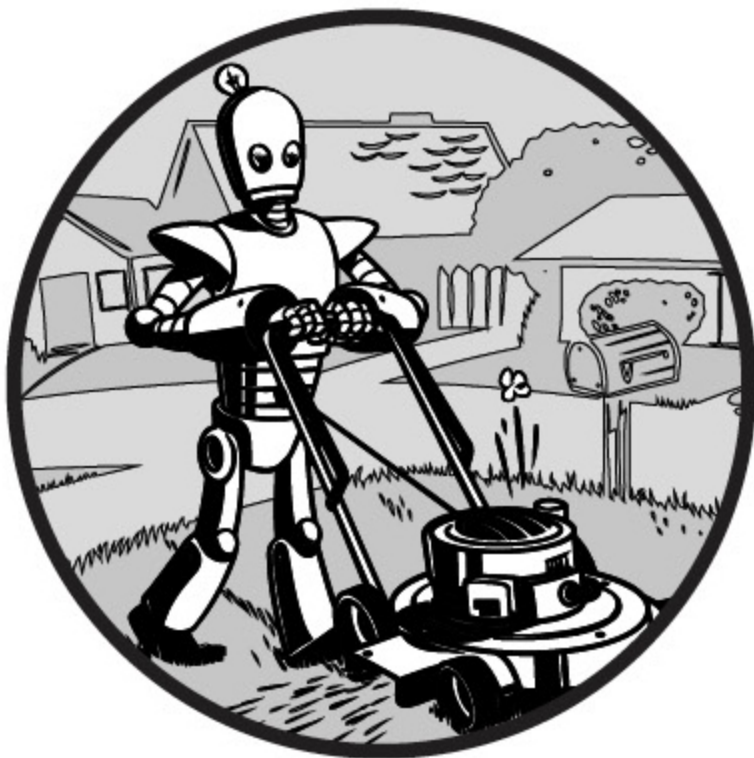
deslizándolas hacia arriba, hacia abajo, hacia la derecha o hacia la izquierda con las flechas de desplazamiento. Puedes conseguir una puntuación bastante alta moviéndolas en direcciones aleatorias. Escribe un programa que abra el juego en <https://play2048.co> y envía pulsaciones de las teclas de dirección para jugar automáticamente.

Verificación de enlaces

Escribe un programa que, dada la URL de una página web, busque todos los enlaces `<a>` de la página y compruebe los resultados en un código de estado “404 Not Found”. El programa debería imprimir cualquier enlace que no funcione.

1. [1](#) La respuesta es no.

Hojas de cálculo de Excel



Aunque a menudo no pensemos en las hojas de cálculo como herramientas de programación, casi todo el mundo las usa para organizar información en estructuras de datos bidimensionales, hacer cálculos con fórmulas y producir gráficos. En los dos próximos capítulos, vamos a integrar Python en dos aplicaciones de hojas de cálculo populares: Microsoft Excel y Google Sheets.

Excel es una aplicación de hojas de cálculo popular y potente. El módulo `openpyxl` permite que nuestros programas de Python lean y modifiquen archivos de hojas de cálculo de Excel. Por ejemplo, imagina que te encargan la tediosa tarea de copiar datos de una

hoja y pegarlos en otra. O que tienes que pasar por miles de filas y seleccionar solo unas pocas para hacer pequeñas ediciones a partir de unos criterios. O que tienes que mirar cientos de hojas de presupuestos de departamentos buscando cualquiera que esté en números rojos. Ese es justo el tipo de tarea aburrida y mecánica que Python puede hacer por ti.

Aunque Excel es software propietario de Microsoft, LibreOffice es una alternativa gratuita que funciona en Windows, macOS y Linux. LibreOffice Calc utiliza el formato de archivo .xlsx de Excel para las hojas de cálculo, lo que significa que el módulo `openpyxl` también puede trabajar con hojas de esta aplicación. Puedes descargarla en <https://www.libreoffice.org>. Incluso aunque ya tengas Excel instalado en tu ordenador, es posible que este programa te resulte más fácil de usar. No obstante, todas las capturas de pantalla de este capítulo son de la versión en la nube de Office 365 Excel.

El módulo `openpyxl` funciona con archivos Excel, no con las aplicaciones Excel de escritorio ni para la nube. Si estás usando Office 365 en la nube, haz clic en Archivo > Guardar > Descargar una copia para descargar una hoja de cálculo, ejecuta tu *script* de Python para editar ese archivo y, después, vuelve a subirlo a Office 365 para ver los resultados. Si tienes la versión de escritorio de Excel, cierra la hoja de cálculo, ejecuta el *script* de Python para editar el archivo y vuelve a abrirlo con Excel para ver los cambios.

Python no incluye `openpyxl`, así que tendrás que instalarlo. En el apéndice A encontrarás información sobre cómo instalar paquetes de terceros con la herramienta `pip` de Python. La documentación completa de `openpyxl` se encuentra en <https://openpyxl.readthedocs.io/en/stable/>.

En este capítulo usaremos varios archivos de hojas de cálculo de ejemplo. Puedes descargarlos de la página de materiales del libro.

Leer archivos Excel

Antes de nada, vamos a repasar unas definiciones básicas. Un documento de hojas de cálculo de Excel se denomina “libro”. Un libro se guarda en un archivo con la extensión .xlsx. Cada libro

puede contener múltiples hojas de cálculo, también llamadas hojas, sin más. La hoja que está viendo el usuario en el momento (o en la que estaba al cerrar Excel) es la hoja activa. Cada hoja tiene columnas (designadas con letras a partir de la A) y filas (designadas con números a partir del 1). Una casilla en una fila y columna en particular se denomina “celda”. Cada celda puede contener un valor numérico o de texto. La cuadrícula de celdas y sus datos conforma una hoja.

Los ejemplos de este capítulo utilizan una hoja de cálculo llamada `example3.xlsx` guardada en el directorio de trabajo actual. Puedes crear la hoja tú o descargarla de los recursos en línea del libro. La figura 14.1 muestra las pestañas para las tres hojas del archivo, llamadas Hoja1, Hoja2 y Hoja3, si lo creas tú, o Sheet1, Sheet2 y Sheet3, si lo has descargado.

La primera hoja del archivo de ejemplo debería ser como la tabla 14.1. (Si no has descargado `example3.xlsx`, deberías introducir estos datos en la hoja).

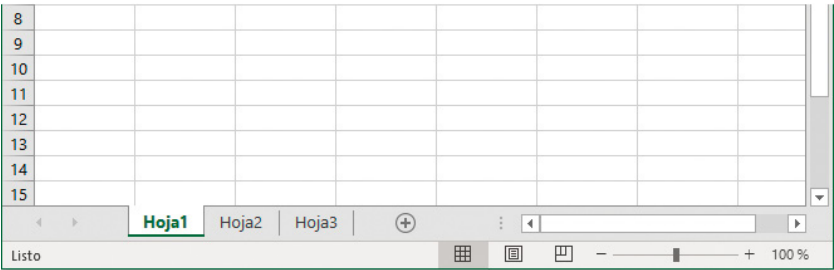


Figura 14.1. Las pestañas de las hojas de un libro están en la esquina inferior izquierda de Excel.

Tabla 14.1. La hoja de cálculo de `example3.xlsx`.

R
Apple 2035 1:34:02 PM
Banana 2035 3:41:23 AM
Orange 2035 12:46:51 PM
Apple 2035 8:59:43 AM
Apple 2035 2:07:00 AM
Banana 2035 6:10:37 PM

28/10/2035 2:40:46 AM

Ahora que ya tenemos nuestra hoja de cálculo de ejemplo, vamos a ver cómo manipularla con el módulo `openpyxl`.

Abrir un libro

Una vez importado el módulo `openpyxl`, podemos abrir archivos `.xlsx` con la función `openpyxl.load_workbook()`. Introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > import openpyxl

> > > wb = openpyxl.load_workbook('example3.xlsx')

> > > type(wb)

<class 'openpyxl.workbook.workbook.Workbook'>
```

La función `openpyxl.load_workbook()` coge el nombre del archivo y devuelve un valor del tipo de datos `Workbook`. Este objeto `Workbook` representa el archivo Excel, un poco igual que un objeto `File` representa un archivo de texto abierto.

Obtener hojas del libro

Podemos obtener una lista de los nombres de todas las hojas del libro accediendo al atributo `sheetnames`. Introduce esto en el intérprete de comandos interactivo:

```
> > > import openpyxl

> > > wb = openpyxl.load_workbook('example3.xlsx')

> > > wb.sheetnames # Los nombres de las hojas del libro

['Sheet1', 'Sheet2', 'Sheet3']

> > > sheet = wb['Sheet3'] # Obtiene una hoja del libro.

> > > sheet
```

```
< Worksheet "Sheet3" >
```

```
> > > type(sheet)
```

```
< class 'openpyxl.worksheet.worksheet.Worksheet' >
```

```
> > > sheet.title # Obtiene el título de la hoja como una cadena.
```

```
'Sheet3'
```

```
> > > another_sheet = wb.active # Obtiene la hoja activa.
```

```
> > > another_sheet
```

```
< Worksheet "Sheet1" >
```

Cada hoja está representada por un objeto Worksheet, que se obtiene usando los corchetes con la cadena del nombre de la hoja, como una clave de diccionario. Finalmente, podemos usar el atributo active de un objeto Workbook para obtener la hoja activa del libro. La hoja activa es la que aparece encima cuando se abre el libro en Excel. Una vez que tenemos el objeto Worksheet, podemos obtener su nombre del atributo title.

Obtener celdas de las hojas

Cuando ya tenemos un objeto Worksheet, podemos obtener un objeto Cell por su nombre. Pon esto en el intérprete de comandos interactivo:

```
> > > import openpyxl
```

```
> > > wb = openpyxl.load_workbook('example3.xlsx')
```

```
> > > sheet = wb['Sheet1'] # Obtiene una hoja de un libro.
```

```
> > > sheet['A1'] # Obtiene una celda de la hoja.
```

```
< Cell 'Sheet1'.A1 >
```

```
> > > sheet['A1'].value # Obtiene el valor de la celda.
```

```
datetime.datetime(2035, 4, 5, 13, 34, 2)
```

```
> > > c = sheet['B1'] # Obtiene otra celda de la hoja.
```

```
> > > c.value
```

```
'Apples'
```

```
> > > # Obtiene la fila, la columna y el valor de la celda.
```

```
> > > f'Row {c.row}, Column {c.column} is {c.value}'
```

```
'Row 1, Column 2 is Apples'
```

```
> > > f'Cell {c.coordinate} is {c.value}'
```

```
'Cell B1 is Apples'
```

```
> > > sheet['C1'].value
```

73

El objeto Cell tiene un atributo value que contiene, para sorpresa de nadie, el valor almacenado en esa celda. También tiene atributos row, column y coordinate que ofrecen información de la ubicación de la celda. Aquí, acceder al atributo value de nuestro objeto Cell para la celda B1 nos da la cadena 'Apples'. El atributo row nos da el entero 1, el atributo column nos da 2, y coordinate, 'B1'. El módulo openpyxl interpretará automáticamente las fechas de la columna A y las devolverá como valores datetime en vez de como cadenas. El capítulo 19 explica en mayor profundidad este tipo de datos.

Especificar una columna por letras puede ser difícil para programar, sobre todo porque, después de Z, las columnas empiezan a usar dos letras: AA, AB, AC, etc. Como alternativa, podemos obtener una celda usando el método cell() y pasándole enteros para sus argumentos de palabra clave row y column. El primer entero para una fila o columna es 1, no 0. Continúa con el ejemplo del intérprete de comandos interactivo escribiendo esto:

```
> > > sheet.cell(row = 1, column = 2)
```

```
<Cell 'Sheet1'.B1 >
```

```
> > > sheet.cell(row = 1, column = 2).value
```

```
'Apples'
```

```
> > > for i in range(1, 8, 2): # Pasa por todas las demás filas.
```

```
... print(i, sheet.cell(row = i, column = 2).value)
```

```
...
```

```
1 Apples
```

```
3 Pears
```

```
5 Apples
```

```
7 Strawberries
```

Usar el método `cell()` de una hoja pasándole `row = 1` y `column = 2` nos da un objeto `Cell` para la celda B1, igual que especificar `sheet['B1']`.

Sirviéndonos de este método `cell()` y sus argumentos de palabra clave, hemos escrito un bucle `for` para imprimir los valores de una serie de celdas. Supongamos que queremos bajar por la columna B e imprimir el valor de todas las celdas con un número de fila impar. Pasando 2 como parámetro de “paso” para la función `range()`, podemos obtener las celdas de cada segunda fila (en este caso, todas las impares). Este ejemplo pasa la variable `i` del bucle `for` como argumento `row` del método `cell()` y utiliza 2 como argumento de `column` en cada llamada al método. Observa que este método acepta el entero 2, no la cadena ‘B’.

Podemos determinar el tamaño de la hoja con los atributos `max_row` y `max_column` del objeto `Worksheet`. Introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > import openpyxl
```

```
> > > wb = openpyxl.load_workbook('example3.xlsx')
```

```
> > > sheet = wb['Sheet1']
```

```
> > > sheet.max_row # Obtiene el número de fila más alto.
```

```
7
```

```
> > > sheet.max_column # Obtiene el número de columna más alto.
```

```
3
```

Fíjate en que el atributo `max_column` es un entero y no la letra que aparece en Excel.

Conversión entre letras y números de columna

Para convertir números en letras, llamaremos a la función `openpyxl.utils.get_column_letter()`. Para pasar de letras a números, llamaremos a la función `openpyxl.utils.column_index_from_string()`. Introduce esto en el intérprete de comandos interactivo:

```
> > > import openpyxl
```

```
> > > from openpyxl.utils import get_column_letter,  
column_index_from_string
```

```
> > > get_column_letter(1) # Traduce la columna 1 a una letra.
```

```
'A'
```

```
> > > get_column_letter(2)
```

```
'B'
```

```
> > > get_column_letter(27)
```

```
'AA'
```

```
> > > get_column_letter(900)
```

```
'AHP'
```

```

> > > wb = openpyxl.load_workbook('example3.xlsx')

> > > sheet = wb['Sheet1']

> > > get_column_letter(sheet.max_column)

'C'

> > > column_index_from_string('A') # Obtiene el número de A.

1

> > > column_index_from_string('AA')

27

```

Tras importar estas dos funciones del módulo openpyxl.utils, podemos llamar a get_column_letter() y pasarle un entero como 27 para averiguar el nombre de la 27ª columna. La función column_index_from_string() hace lo contrario: le pasamos la letra de una columna y nos dice el número. No hace falta tener un libro cargado para usar estas funciones.

Obtener filas y columnas

Podemos hacer rebanadas con objetos Worksheet para obtener todos los objetos Cell de una fila, columna o área rectangular de la hoja de cálculo. Después, podemos pasar en bucle por todas las celdas de la rebanada. Escribe esto en el intérprete de comandos interactivo:

```

> > > import openpyxl

> > > wb = openpyxl.load_workbook('example3.xlsx')

> > > sheet = wb['Sheet1']

> > > sheet['A1':'C3'] # Obtiene las celdas de A1 a C3.

((<Cell 'Sheet1'.A1>, <Cell 'Sheet1'.B1>, <Cell 'Sheet1'.C1>),
 (<Cell 'Sheet1'.A2>, <Cell

```

```
'Sheet1'.B2>, <Cell 'Sheet1'.C2>), (<Cell 'Sheet1'.A3>, <Cell  
'Sheet1'.B3>, <Cell
```

```
'Sheet1'.C3>))
```

```
> > > for row_of_cell_objects in sheet['A1':'C3']: 1
```

```
... for cell_obj in row_of_cell_objects: 2
```

```
. print(cell_obj.coordinate, cell_obj.value)
```

```
... print('--- END OF ROW )
```

```
...
```

```
A1 2035-04-05 13:34:02
```

```
B1 Apples
```

```
C1 73
```

```
--- END OF ROW ---
```

```
A2 2035-04-05 03:41:23
```

```
B2 Cherries
```

```
C2 85
```

```
--- END OF ROW ---
```

```
A3 2035-04-06 12:46:51
```

```
B3 Pears
```

```
C3 14
```

```
--- END OF ROW ---
```

Aquí, especificamos ['A1':'C3'] para obtener una rebanada de los objetos Cell del área que va de A1 a C3 y obtenemos una tupla que contiene los objetos Cell de esa área.

Esta tupla contiene tres tuplas: una por cada fila de arriba abajo. Cada una de estas tres tuplas internas contiene los objetos Cell de una fila del área deseada, de izquierda a derecha. Así pues, en total, nuestra rebanada de esta hoja contiene todos los objetos Cell que hay en el área de A1 a C3, empezando desde la celda de la esquina superior izquierda y terminando en la de la esquina inferior derecha.

Para imprimir los valores de cada celda de esa área, usamos dos bucles for. El más externo pasa por cada fila de la rebanada 1. Después, para cada fila, el bucle for anidado pasa por cada celda de la fila 2.

Para acceder a los valores de celdas en una fila o columna en concreto, también podemos usar los atributos rows y columns del objeto Worksheet. Estos atributos deben convertirse a listas con la función list() para poder usar los corchetes y un índice con ellos. Introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > import openpyxl
```

```
> > > wb = openpyxl.load_workbook('example3.xlsx')
```

```
> > > sheet = wb['Sheet1']
```

```
> > > list(sheet.columns)[1] # Obtiene las celdas de la segunda columna.
```

```
(<Cell 'Sheet1'.B1>, <Cell 'Sheet1'.B2>, <Cell 'Sheet1'.B3>,  
<Cell 'Sheet1'.B4>, <Cell
```

```
'Sheet1'.B5>, <Cell 'Sheet1'.B6>, <Cell 'Sheet1'.B7>)
```

```
> > > for cell_obj in list(sheet.columns)[1]:
```

```
... print(cell_obj.value)
```

```
.
```

```
Apples
```

```
Cherries
```


Pears

Oranges

Apples

Bananas

Strawberries

Usar el atributo `rows` de un objeto `Worksheet` pasado a `list()` nos dará una lista de tuplas. Cada una de estas tuplas representa una fila y contiene los objetos `Cell` de esa fila. El atributo `columns`, pasado a `list()`, también nos da una lista de tuplas, cada una de ellas con los objetos `Cell` de una columna en particular. Para `example3.xlsx`, como hay siete filas y tres columnas, `list(sheet.rows)` nos da una lista de siete tuplas (cada una con tres objetos `Cell`), y `list(sheet.columns)` nos da una lista de tres tuplas (cada una con siete objetos `Cell`).

Para acceder a una tupla en concreto, podemos referirnos a ella por su índice en la tupla superior. Por ejemplo, para obtener la tupla que representa la columna B, usaríamos `list(sheet.columns)[1]`. Para obtener la tupla que contiene los objetos `Cell` de la columna A, usaríamos `list(sheet.columns)[0]`. Una vez que tenemos una tupla que representa una fila o columna, podemos pasar en bucle por sus objetos `Cell` para imprimir sus valores.

Repaso de libros, hojas y celdas

Como repaso rápido, aquí tienes un resumen de todas las funciones, métodos y tipos de datos implicados en la lectura de celdas de una hoja de cálculo:

1. Importa el módulo `openpyxl`.
2. Llama a la función `openpyxl.load_workbook()` para obtener un objeto `Workbook`.
3. Usa el atributo `active` o `sheetnames`.
4. Obtén un objeto `Worksheet`.

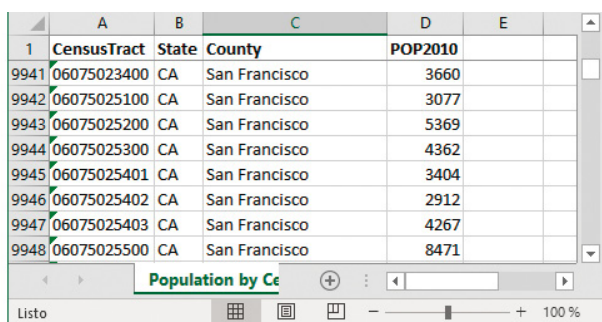
5. Usa el indexado o el método `cell()` de la hoja con argumentos de palabra clave `row` y `column`.

6. Obtén un objeto `Cell`.

7. Lee el atributo `value` del objeto `Cell`.

Proyecto 9: Reunir estadísticas censales

Supongamos que tienes una hoja de cálculo con datos del censo de EE. UU. de 2010 y se te ha encomendado la aburrida tarea de pasar por sus miles de filas para contar la población y el número de secciones censales para cada condado. (Una sección censal es simplemente un área geográfica definida a efectos del censo). Cada fila representa una sola sección censal. Llamaremos al archivo `censuspopdata.xlsx`; puedes descargarlo en los recursos en línea de este libro. La figura 14.2 muestra su contenido.



	A	B	C	D	E
1	CensusTract	State	County	POP2010	
9941	06075023400	CA	San Francisco	3660	
9942	06075025100	CA	San Francisco	3077	
9943	06075025200	CA	San Francisco	5369	
9944	06075025300	CA	San Francisco	4362	
9945	06075025401	CA	San Francisco	3404	
9946	06075025402	CA	San Francisco	2912	
9947	06075025403	CA	San Francisco	4267	
9948	06075025500	CA	San Francisco	8471	

Figura 14.2. La hoja de cálculo `censuspopdata.xlsx`.

Incluso aunque Excel pueda calcular automáticamente la suma de las celdas seleccionadas, tienes que seleccionar primero las celdas para cada uno de los más de tres mil condados. Y, aunque solo se tarde unos segundos en calcular la población de un condado a mano, hacerlo para toda la hoja de cálculo llevaría horas.

En este proyecto, vamos a escribir un *script* que pueda leer de la hoja de cálculo del censo para calcular estadísticas de cada condado en cuestión de segundos.

Esto es lo que hace el programa:

- Leer los datos de la hoja de cálculo de Excel.
- Contar el número de secciones censales en cada condado.
- Contar la población total de cada condado.
- Imprimir los resultados.

Esto significa que el código tendrá que hacer lo siguiente:

- Abrir un archivo Excel y leer las celdas con el módulo openpyxl.
- Calcular todos los datos de las secciones y la población y guardarlos en una estructura de datos.
- Escribir la estructura de datos en un archivo de texto con la extensión .py usando el módulo pprint para poder importarlo luego.

Paso 1: leer los datos de la hoja de cálculo

Solo hay una hoja en el libro censuspodata.xlsx, llamada 'Population by Census Tract', y cada fila de la hoja contiene los datos para una sola sección censal. Las columnas son el número de sección (A), la abreviatura del estado (B), el nombre del condado (C) y la población de la sección (D). Abre una nueva pestaña en el editor de archivos e introduce el siguiente código; guarda el archivo como readCensusExcel.py:

```
# readCensusExcel.py - Tabula población de condado y secciones censales
```

```
1 import openpyxl, pprint
```

```
print('Opening workbook...')
```

```
2 wb = openpyxl.load_workbook('censuspodata.xlsx')
```

```
3 sheet = wb['Population by Census Tract']
```

```
county_data = {}
```

```
# PENDIENTE: Rellenar county_data con la población y las
```

secciones de cada condado.

```
print('Reading rows...')
```

```
4 for row in range(2, sheet.max_row + 1):
```

```
# Cada fila de la hoja de cálculo tiene datos para una sección censal.
```

```
state = sheet['B' + str(row)].value
```

```
county = sheet['C' + str(row)].value
```

```
pop = sheet['D' + str(row)].value
```

```
# PENDIENTE: Abrir un nuevo archivo y escribir en él el contenido de county_data.
```

Este código importa el módulo `openpyxl`, así como el módulo `pprint` que usaremos para imprimir los datos definitivos del condado 1. A continuación, abre el archivo `censuspodata.xlsx` 2, obtiene la hoja con los datos del censo 3 y empieza a iterar por sus filas 4. Observa que también hemos creado una variable llamada `county_data`, que contendrá la población y el número de secciones calculadas para cada condado. Pero, antes de poder almacenar nada en ella, hay que determinar exactamente cómo estructurar los datos dentro.

Paso 2: poblar la estructura de datos

En Estados Unidos, los estados tienen abreviaturas de dos letras y se dividen en condados. La estructura de datos almacenada en `county_data` será un diccionario con las abreviaturas de los estados como claves. Cada abreviatura conducirá a otro diccionario, cuyas claves serán las cadenas de los nombres de condado del estado en cuestión. A su vez, cada nombre de condado dirigirá a un diccionario con solo dos claves, `'tracts'` y `'pop'`. Estas claves llevarán al número de secciones censales y la población de cada condado. A modo de ejemplo, el diccionario sería así:

```
{ 'AK': { 'Aleutians East': { 'pop': 3141, 'tracts': 1 },
```

‘Aleutians West’: {‘pop’: 5561, ‘tracts’: 2},

‘Anchorage’: {‘pop’: 291826, ‘tracts’: 55},

‘Bethel’: {‘pop’: 17013, ‘tracts’: 3},

‘Bristol Bay’: {‘pop’: 997, ‘tracts’: 1},

--*fragmento omitido*--

Si el diccionario anterior estuviese almacenado en `county_data`, las siguientes expresiones se evaluarían así:

```
> > > county_data[‘AK’][‘Anchorage’][‘pop’]
```

```
291826
```

```
> > > county_data[‘AK’][‘Anchorage’][‘tracts’]
```

```
55
```

En general, las claves del diccionario `county_data` tienen este aspecto:

```
county_data[state abbrev][county][‘tracts’]
```

```
county_data[state abbrev][county][‘pop’]
```

Ahora que sabemos cómo estructurar `county_data`, podemos escribir el código que lo rellene con los datos de los condados. Añade el siguiente código al final del programa:

```
# readCensusExcel.py - Tabula población de condado y secciones censales
```

--*fragmento omitido*--

```
for row in range(2, sheet.max_row + 1):
```

```
# Cada fila de la hoja de cálculo tiene datos para una sección censal.
```

```
state = sheet[‘B’ + str(row)].value
```

```
county = sheet['C' + str(row)].value
```

```
pop = sheet['D' + str(row)].value
```

```
# Se asegura de que la clave para este estado existe.
```

```
1 county_data.setdefault(state, )
```

```
# Se asegura de que la clave para este condado del estado  
existe.
```

```
2 county_data[state].setdefault(county, {'tracts': 0, 'pop': 0})
```

```
# Cada fila representa una sección censal, así que suma uno.
```

```
3 county_data[state][county]['tracts'] = 1
```

```
# Aumenta la población del condado con la población de esta  
sección censal.
```

```
4 county_data[state][county]['pop'] = int(pop)
```

```
# PENDIENTE: Abrir un nuevo archivo y escribir en él el contenido  
de county_data.
```

Las dos últimas líneas de código hacen el trabajo de cálculo real, aumentando el valor de tracts 3 e incrementando el valor de pop 4 para el condado actual en cada iteración del bucle for.

El resto del código está ahí porque no se puede añadir un diccionario de condado como valor para una clave de abreviatura de estado hasta que exista esa clave en county_data. (Es decir, county_data['AK']['Anchorage']['tracts'] + = 1 provocará un error si la clave 'AK' todavía no existe). Para asegurarnos de que las claves de las abreviaturas de estado existen en nuestra estructura de datos, tenemos que llamar al método setdefault() para establecer un valor si no hay todavía uno para state 1.

Igual que el diccionario county_data necesita un diccionario como el valor de cada clave de abreviatura, cada uno de esos diccionarios necesitará su propio diccionario como valor para cada clave de condado 2. Y cada uno de esos diccionarios necesitará a su vez las

claves 'tracts' y 'pop' que empiezan con el valor entero 0. (Si en algún momento te pierdes en la estructura de datos, vuelve a mirar el diccionario de ejemplo al principio de este apartado). Dado que setdefault() no hará nada si la clave ya existe, podemos llamarlo en cada iteración del bucle for sin problemas.

Paso 3: escribir los resultados en un archivo

Una vez que haya terminado el bucle for, el diccionario county_data contendrá toda la información sobre la población y las secciones censales organizada por condado y estado. Llegados a este punto, podemos programar más para escribir estos datos en un archivo de texto o en otra hoja de cálculo de Excel. De momento, solo vamos a usar la función pprint.pformat() para escribir el valor del diccionario county_data como cadena masiva en un archivo llamado census2010.py. Añade el siguiente código al final del programa (asegurándote de mantenerlo sin sangrado para que se quede fuera del bucle):

```
# readCensusExcel.py - Tabula población de condado y secciones censales.
```

```
--fragmento omitido--
```

```
# Abre un nuevo archivo y escribe en él el contenido de county_data.
```

```
print('Writing results...')
```

```
result_file = open('census2010.py', 'w')
```

```
result_file.write('allData = ' + pprint.pformat(county_data))
```

```
result_file.close()
```

```
print('Done.')
```

La función pprint.pformat() produce una cadena formateada como código Python válido. Al sacarla a un archivo llamado census2010.py, ¡hemos generado un programa de Python para nuestro programa de Python! Puede parecer complicado, pero la

ventaja es que ahora podemos importar census2010.py como cualquier otro módulo de Python. En el intérprete de comandos interactivo, cambia el directorio de trabajo actual a la carpeta donde está el archivo census2010.py recién creado e impórtalo:

```
> > > import census2010
```

```
> > > census2010.allData['AK'] ['Anchorage']
```

```
{'pop': 291826, 'tracts': 55}
```

```
> > > anchorage_pop = census2010.allData['AK']  
['Anchorage'] ['pop']
```

```
> > > print('The 2010 population of Anchorage was ' +  
str(anchorage_pop))
```

The 2010 population of Anchorage was 291826

El programa readCensusExcel.py era código de usar y tirar: una vez que tenemos los resultados guardados en census2010.py, no necesitamos volver a ejecutar el programa. Siempre que necesitemos los datos de los condados, podemos simplemente ejecutar `run census2010`.

Calcular estos datos a mano habría llevado horas, pero este programa lo ha hecho en segundos. Con `openpyxl`, no tendrás problemas para extraer información guardada en una hoja de Excel y realizar cálculos con ella. Puedes descargar el programa completo en los recursos en línea del libro.

Ideas para programas similares

Muchos negocios y oficinas usan Excel para almacenar varios tipos de datos y no es raro que las hojas crezcan descontroladamente. Cualquier programa que analice una hoja de cálculo de Excel tendrá una estructura similar: carga el archivo, prepara algunas variables o estructuras de datos y pasa en bucle por cada una de las filas de la hoja de cálculo. Un programa así podría hacer lo siguiente:

- Comparar datos entre varias filas de una hoja.

- Abrir varios archivos de Excel y comparar datos entre hojas.
- Comprobar si una hoja tiene filas en blanco o datos no válidos en alguna celda y alertar al usuario si es el caso.
- Leer datos de una hoja de cálculo y usarlos como entrada para tus programas de Python.

Escribir en archivos de Excel

El módulo `openpyxl` también ofrece maneras de escribir datos, lo que significa que nuestros programas pueden crear y editar archivos de hoja de cálculo. Con Python, es fácil crear hojas con miles de filas.

Crear y guardar archivos de Excel

Llamaremos a la función `openpyxl.Workbook()` para crear un nuevo objeto `Workbook` en blanco. Introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > import openpyxl

> > > wb = openpyxl.Workbook() # Crea un libro en blanco.

> > > wb.sheetnames # El libro empieza con una hoja.

['Sheet']

> > > sheet = wb.active

> > > sheet.title

'Sheet'

> > > sheet.title = 'Spam Bacon Eggs Sheet' # Cambia el título.

> > > wb.sheetnames

['Spam Bacon Eggs Sheet']
```

El libro empezará con una sola hoja llamada `Sheet`. Podemos

cambiar ese nombre almacenando una nueva cadena en su atributo `title`.

Cada vez que modifiquemos el objeto `Workbook` o sus hojas y celdas, no se guardará el archivo hasta que llamemos al método `save()`. Escribe lo siguiente en el intérprete de comandos interactivo (con `example3.xlsx` en el directorio de trabajo actual):

```
> > > import openpyxl

> > > wb = openpyxl.load_workbook('example3.xlsx')

> > > sheet = wb['Sheet1']

> > > sheet.title = 'Spam Spam Spam'

> > > wb.save('example3_copy.xlsx') # Guarda el libro.
```

Aquí cambiamos el nombre de nuestra hoja. Para guardar el cambio, pasamos un nombre de archivo como cadena al método `save()`. Pasar un nombre de archivo distinto del original, como `'example3_copy.xlsx'`, guarda los cambios en una copia de la hoja de cálculo.

Siempre que edites una hoja de cálculo cargada desde un archivo, deberías guardar la nueva hoja editada con un nombre distinto del original. De ese modo, conservarás el archivo original por si un fallo en el código hace que el archivo nuevo contenga datos incorrectos o corrompidos. Además, el método `save()` no funcionará si la hoja de cálculo está en ese momento abierta en la aplicación de escritorio Excel. Primero, hay que cerrar la hoja y después ejecutar el programa de Python.

Crear y eliminar hojas

Podemos crear o eliminar hojas en un libro con el método `create_sheet()` y el operador `del`. Introduce esto en el intérprete de comandos interactivo:

```
> > > import openpyxl
```

```
> > > wb = openpyxl.Workbook()
```

```
> > > wb.sheetnames
```

```
['Sheet']
```

```
> > > wb.create_sheet() # Añade una hoja.
```

```
<Worksheet "Sheet1">
```

```
> > > wb.sheetnames
```

```
['Sheet', 'Sheet1']
```

```
> > > # Crea una nueva hoja en el índice 0.
```

```
> > > wb.create_sheet(index=0, title='First Sheet')
```

```
<Worksheet "First Sheet">
```

```
> > > wb.sheetnames
```

```
['First Sheet', 'Sheet', 'Sheet1']
```

```
> > > wb.create_sheet(index=2, title='Middle Sheet')
```

```
<Worksheet "Middle Sheet">
```

```
> > > wb.sheetnames
```

```
['First Sheet', 'Sheet', 'Middle Sheet', 'Sheet1']
```

El método `create_sheet()` devuelve un nuevo objeto `Worksheet` llamado `SheetX`, que es por defecto la última hoja del libro. Como opción, podemos especificar el índice y el nombre de la hoja nueva con los argumentos `index` y `title`.

Continúa con el ejemplo anterior introduciendo esto:

```
> > > wb.sheetnames
```

```
['First Sheet', 'Sheet', 'Middle Sheet', 'Sheet1']
```

```
> > > del wb ['Middle Sheet']
```

```
> > > del wb ['Sheet1']
```

```
> > > wb.sheetnames
```

```
['First Sheet', 'Sheet']
```

Podemos usar el operador `del` para borrar una hoja de un libro, igual que podíamos usarlo para eliminar un par clave-valor de un diccionario.

Recuerda llamar al método `save()` para guardar los cambios después de añadir o quitar hojas en el libro.

Escribir valores en celdas

Escribir valores en celdas se parece mucho a escribir valores en claves de diccionario. Introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > import openpyxl
```

```
> > > wb = openpyxl.Workbook()
```

```
> > > sheet = wb['Sheet']
```

```
> > > sheet['A1'] = 'Hello, world!' # Edita el valor de la celda.
```

```
> > > sheet['A1'].value
```

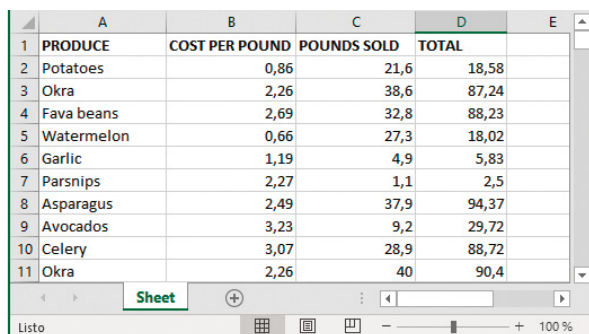
```
'Hello, world!'
```

Si tenemos las coordenadas de la celda como una cadena, podemos usarlas como una clave de diccionario en el objeto `Worksheet` para especificar en qué celda hay que escribir.

Proyecto 10: Actualizar una hoja de cálculo

En este proyecto, vamos a escribir un programa de cálculo para actualizar celdas en una hoja de cálculo de venta de productos frescos. El programa pasará por la hoja de cálculo, buscará tipos de

productos específicos y actualizará sus precios. Descarga la hoja produceSales3.xlsx en los recursos en línea del libro. La figura 14.3 muestra el aspecto de la hoja de cálculo.



	A	B	C	D	E
	PRODUCE	COST PER POUND	POUNDS SOLD	TOTAL	
2	Potatoes	0,86	21,6	18,58	
3	Okra	2,26	38,6	87,24	
4	Fava beans	2,69	32,8	88,23	
5	Watermelon	0,66	27,3	18,02	
6	Garlic	1,19	4,9	5,83	
7	Parsnips	2,27	1,1	2,5	
8	Asparagus	2,49	37,9	94,37	
9	Avocados	3,23	9,2	29,72	
10	Celery	3,07	28,9	88,72	
11	Okra	2,26	40	90,4	

Figura 14.3. Una hoja de cálculo de venta de productos frescos.

Cada fila representa una venta individual. Las columnas son el tipo de producto vendido (A), el precio por libra de ese producto (B), el número de libras vendidas (C) y los ingresos totales por esa venta (D). La columna TOTAL contiene una fórmula de Excel de tipo $=\text{ROUND}(B2 * C2, 2)$, que multiplica el precio por libra por las libras vendidas en la misma fila y redondea el resultado al céntimo más próximo. Con esta fórmula, las celdas de la columna TOTAL se actualizarán solas si hay algún cambio en las columnas COST PER POUND y POUNDS SOLD.

Ahora imagina que los precios del ajo, el apio y los limones se hubiesen metido mal, dejándonos con la tediosa tarea de repasar miles de filas en esta hoja de cálculo para actualizar el precio por unidad de cualquier fila con esos productos. No podemos simplemente hacer una búsqueda y reemplazar el precio porque puede haber otros productos que valgan lo mismo y no queremos “corregirlos” por error. Si hay miles de celdas, hacer esto a mano nos llevaría horas. Pero podemos escribir un programa que lo haga en segundos. El programa debería hacer lo siguiente:

- Pasar en bucle por todas las filas.
- Si la fila es de apio, ajo o limones, cambiar el precio.

Eso significa que el código tendrá que:

- Abrir el archivo de hoja de cálculo.
- Para cada fila, comprobar si el valor de la columna A es Celery, Garlic o Lemon.
- Si lo es, actualizar el precio en la columna B.
- Guardar la hoja de cálculo en un archivo nuevo (para no perder la original, por si acaso).

Paso 1: configurar una estructura de datos con la información actualizada

Los precios que hay que actualizar son estos:

- Apio (*celery*): 1,19.
- Ajo (*garlic*): 3,07.
- Limón (*lemon*): 1,27.

Podríamos escribir código para poner esos precios nuevos, así:

```
if produce_name == 'Celery':
```

```
    cell_obj = 1.19
```

```
if produce_name == 'Garlic':
```

```
    cell_obj = 3.07
```

```
if produce_name == 'Lemon':
```

```
    cell_obj = 1.27
```

Pero no es buena idea codificar de una manera tan rígida los productos y el precio actualizado. Si hubiera que volver a actualizar la hoja con precios o productos diferentes, habría que cambiar buena parte del código y, cada vez que cambiamos código, corremos el riesgo de introducir fallos.

Una solución más flexible es guardar la información de precio correcta en un diccionario y escribir código que use esa estructura

de datos. En una pestaña nueva del editor de archivos, escribe el siguiente código:

```
# updateProduce.py - Corrige precios en la hoja de cálculo de venta de productos
```

```
import openpyxl
```

```
wb = openpyxl.load_workbook('produceSales3.xlsx')
```

```
sheet = wb['Sheet']
```

```
# Los tipos de producto y sus precios actualizados
```

```
PRICE_UPDATES = {'Garlic': 3.07,
```

```
'Celery': 1.19,
```

```
'Lemon': 1.27}
```

```
# PENDIENTE: Pasar en bucle por las filas y actualizar los precios.
```

Guarda esto como updateProduce.py. Si necesitásemos volver a actualizar la hoja, solo tendríamos que actualizar el diccionario PRICE_UPDATES, sin tocar más código.

Paso 2: comprobar todas las filas y actualizar los precios incorrectos

La siguiente parte del programa pasará en bucle por todas las filas de la hoja de cálculo. Añade el siguiente código al final de updateProduce.py:

```
# updateProduce.py - Corrige precios en la hoja de cálculo de venta de productos
```

```
--fragmento omitido--
```

```
# Pasa en bucle por las filas y actualiza los precios.
```

```
1 for row_num in range(2, sheet.max_row + 1): # Se salta la primera fila.
```

```
2 produce_name = sheet.cell(row = row_num,  
column = 1).value
```

```
3 if produce_name in PRICE_UPDATES:
```

```
sheet.cell(row = row_num, column = 2).value =  
PRICE_UPDATES[produce_name]
```

```
4 wb.save('updatedProduceSales3.xlsx')
```

Pasamos en bucle por las filas a partir de la 2, ya que la 1 es solo el encabezado 1. La celda de la columna 1 (esto es, la columna A) se almacenará en la variable `produce_name` 2. Si `produce_name` existe como clave en el diccionario `PRICE_UPDATES` 3, sabemos que hay que corregir el precio en esa fila. El precio correcto estará en `PRICE_UPDATES[produce_name]`. Fíjate en lo limpio que deja el código usar `PRICE_UPDATES`. Solo utiliza una declaración `if`, en lugar de una línea aparte como `if produce_name == 'Garlic':` para cada tipo de producto que hay que actualizar. Y, dado que el código usa el diccionario `PRICE_UPDATES` en vez de hacer una codificación rígida de nombres y precios actualizados en el bucle `for`, podemos modificar solo `PRICE_UPDATES`, sin tocar el resto del código, si la hoja requiere cambios adicionales.

Después de pasar por toda la hoja de cálculo y hacer cambios, el código guarda el objeto `Workbook` en `updatedProduceSales3.xlsx` 4. No sobrescribe la antigua hoja de cálculo por si hay algún fallo en el programa y la hoja actualizada está mal. Una vez comprobado que la hoja actualizada está bien, podemos borrar la vieja.

Ideas para programas similares

Puesto que muchos trabajadores emplean hojas de cálculo de Excel constantemente, un programa que pueda editar y escribir archivos de Excel automáticamente podría serles muy útil. Un programa así podría hacer cosas como estas:

- Leer datos de una hoja de cálculo y escribirlos en partes de otra hoja.
- Leer datos de sitios web, archivos de texto o el portapapeles y

escribirlos en una hoja de cálculo.

•“Limpiar” automáticamente datos en una hoja de cálculo. Por ejemplo, podría usar expresiones regulares para leer varios formatos de número de teléfono y editarlos para dejar todos en un solo formato estándar.

Establecer el estilo de fuente de las celdas

Aplicar estilos a determinadas celdas, filas o columnas puede ayudarnos a destacar áreas importantes en nuestra hoja de cálculo. En la de productos frescos, por ejemplo, el programa podrá aplicar negrita a las filas con patata, ajo y chirivía. O tal vez nos interese poner en cursiva todas las filas cuyo peso por libra supere los cinco dólares. Aplicar estilos a partes de una hoja de cálculo grande a mano puede ser muy tedioso, pero nuestro programa podrá hacerlo al instante.

Para personalizar los estilos de fuente en las celdas, importa la función `Font()` del módulo `openpyxl.styles`:

```
from openpyxl.styles import Font
```

Importar la función de esta manera nos permite escribir `Font()` en vez de `openpyxl.styles.Font()`. (Consulta el apartado “Importación de módulos” del capítulo 3 para más información).

El siguiente ejemplo crea un libro nuevo y pone la celda A1 en fuente cursiva a 24 puntos:

```
> > > import openpyxl

> > > from openpyxl.styles import Font

> > > wb = openpyxl.Workbook()

> > > sheet = wb['Sheet']

1 > > > italic_24_font = Font(size=24, italic=True)

2 > > > sheet['A1'].font = italic_24_font
```

```
> > > sheet['A1'] = 'Hello, world!'
```

```
> > > wb.save('styles3.xlsx')
```

En este ejemplo, `Font(size=24, italic=True)` devuelve un objeto `Font`, que almacenamos en `italic_24_font` 1. Los argumentos de palabra clave para `Font()` configuran la información de estilo del objeto y asignar a `sheet['A1'].font` el objeto `italic_24_font` 2 aplica ese estilo de fuente a la información de la celda A1.

Para establecer atributos font, pasa argumentos de palabra clave a `Font()`. La tabla 14.2 recoge los argumentos posibles para la función `Font()`.

Tabla 14.2. Argumentos para objetos `Font`.

Argumentos

FontName Nombre de la fuente, como 'Calibri' o 'Times New Roman'.

FontSize Tamaño en puntos.

FontWeight Fuente en negrita.

FontStyle Fuente cursiva.

Podemos llamar a `Font()` para crear un objeto `Font` y almacenarlo en una variable. Después, asignamos esa variable al atributo `font` de un objeto `Cell`. Por ejemplo, este código crea varios estilos de fuente:

```
> > > import openpyxl
```

```
> > > from openpyxl.styles import Font
```

```
> > > wb = openpyxl.Workbook()
```

```
> > > sheet = wb['Sheet']
```

```
> > > bold_font = Font(name='Times New Roman',  
bold=True)
```

```
> > > sheet['A1'].font = bold_font
```

```
> > > sheet['A1'] = 'Bold Times New Roman'
```

```
> > > italic_font = Font(size=24, italic=True)

> > > sheet['B3'].font = italic_font

> > > sheet['B3'] = '24 pt Italic'

> > > wb.save('styles3.xlsx')
```

Aquí, almacenamos un objeto Font en `bold_font` y establecemos el atributo `font` del objeto Cell de A1 como `bold_font`. Repetimos el proceso con otro objeto Font para establecer la fuente de una segunda celda. Después de ejecutar este código, los estilos de las celdas A1 y B3 de la hoja de cálculo estarán personalizados, como en la figura 14.4.

	A	B	C	D
1	Bold Times New Roman			
2				
3		<i>24 pt Italic</i>		
4				
5				

Figura 14.4. Una hoja de cálculo con estilos de fuente personalizados.

Para la celda A1, establecemos el nombre de la fuente en ‘Times New Roman’ y bold en True para que el texto aparezca en Times New Roman negrita. No especificamos un tamaño en puntos para que el texto use el predeterminado de openpyxl, que es 11. En la celda B3, el texto está en cursiva, con un tamaño de 24 puntos. No especificamos nombre de fuente, así que el texto usa la predeterminada de openpyxl, Calibri.

Fórmulas

Las fórmulas de Excel, que empiezan con un signo de igualdad, pueden configurar celdas para contener valores calculados a partir de otras celdas. En este apartado, utilizaremos el módulo openpyxl para añadir programáticamente fórmulas a celdas, igual que añadiríamos cualquier valor. He aquí un ejemplo:

```
> > > sheet['B9'] = '=SUM(B1:B8)'
```

Este código almacenará la fórmula =SUM(B1:B8) en B9, estableciendo el valor de la celda en la suma de los valores de las celdas de B1 a B8. Puedes ver esto en acción en la figura 14.5.

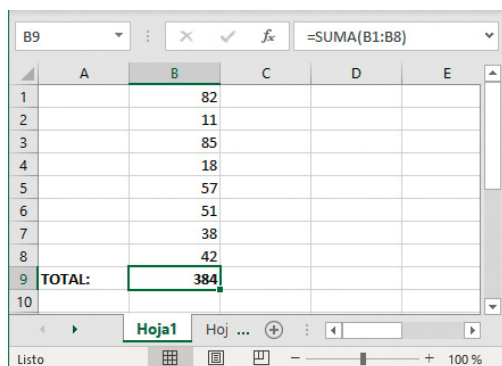


Figura 14.5. La celda B9 contiene la fórmula para sumar los valores de las celdas B1-B8.

Podemos establecer una fórmula de Excel como cualquier otro valor de texto en una celda. Por ejemplo, introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > import openpyxl

> > > wb = openpyxl.Workbook()

> > > sheet = wb['Sheet']

> > > sheet['A1'] = 200

> > > sheet['A2'] = 300

> > > sheet['A3'] = '=SUM(A1:A2)' # Establece la fórmula.

> > > wb.save('writeFormula3.xlsx')
```

Las celdas A1 y A2 están establecidas en 200 y 300, respectivamente. El valor de la celda A3 es una fórmula que suma los valores de A1 y A2. Cuando la hoja de cálculo se abra en Excel, A3 mostrará su valor como 500. El módulo openpyxl no tiene la

capacidad de calcular fórmulas de Excel y poblar celdas con los resultados. Sin embargo, si abrimos este archivo `writeFormula3.xlsx` en Excel, el propio Excel rellenará las celdas con el resultado de la fórmula. Podemos guardar el archivo en Excel y, después, abrirlo pasando el argumento `data_only=True` a `openpyxl.load_workbook()`; los valores de las celdas deberían mostrar los resultados del cálculo en vez de la cadena de la fórmula:

```
> > > # Asegúrate de abrir con writeFormula3.xlsx en Excel y
guardar primero.
```

```
> > > import openpyxl
```

```
> > > wb = openpyxl.load_workbook('writeFormula3.xlsx') #
Abre sin data_only.
```

```
> > > wb.active['A3'].value # Obtiene la cadena de la fórmula.
```

```
'=SUM(A1:A2)'
```

```
> > > wb = openpyxl.load_workbook('writeFormula3.xlsx',
data_only=True) # Abre con data_only.
```

```
> > > wb.active['A3'].value # Obtiene el resultado de la
fórmula.
```

500

De nuevo, solo veremos el resultado 500 en el archivo de la hoja de cálculo si lo abrimos y guardamos en Excel para que la aplicación pueda ejecutar el cálculo de la forma y guardar el resultado en la hoja de cálculo. Este es el valor que lee `openpyxl` cuando pasamos `data_only=True` a `openpyxl.load_workbook()`.

Las fórmulas de Excel ofrecen un nivel de programabilidad para las hojas de cálculo, pero enseguida pueden volverse inmanejables para tareas complicadas. Por ejemplo, aunque estés muy familiarizado con las fórmulas de Excel, es un quebradero de cabeza descifrar qué hace `=IFERROR(TRIM(IF(LEN(VLOOKUP(F7, Sheet2!$A$1:$B$10000, 2, FALSE))>0,SUBSTITUTE(VLOOKUP(F7, Sheet2!$A$1:$B$10000, 2, FALSE), " ", ""))), "")`. El código de Python es mucho más legible.

Ajustar filas y columnas

En Excel, ajustar el tamaño de filas y columnas es tan fácil como hacer clic y arrastrar los bordes del encabezado de la fila o la columna. Pero, si hay que establecer el tamaño de una fila o columna basándose en el contenido de sus celdas, o si necesitamos configurar tamaños en un gran número de archivos de hojas de cálculo, es mucho más rápido escribir un programa de Python que lo haga.

También podemos ocultar filas y columnas o “congelarlas” para que queden siempre visibles en pantalla, apareciendo en cada página al imprimir la hoja (lo cual es útil para los encabezados).

Establecer la altura de fila y el ancho de columna

Un objeto `Worksheet` cuenta con atributos `row_dimensions` y `column_dimensions` que controlan la altura de las filas y la anchura de las columnas. Para ver un ejemplo, introduce esto en el intérprete de comandos interactivo:

```
> > > import openpyxl

> > > wb = openpyxl.Workbook()

> > > sheet = wb['Sheet']

> > > sheet['A1'] = 'Tall row'

> > > sheet['B2'] = 'Wide column'

> > > sheet.row_dimensions[1].height = 70

> > > sheet.column_dimensions['B'].width = 20

> > > wb.save('dimensions3.xlsx')
```

Los atributos `row_dimensions` y `column_dimensions` de una hoja son como valores de diccionario; `row_dimensions` contiene objetos `RowDimension` y `column_dimensions` contiene objetos `ColumnDimension`. En `row_dimensions`, podemos acceder a uno de

los objetos utilizando el número de la fila (en este caso, 1 o 2). En `column_dimensions`, podemos acceder a los objetos usando la letra de la columna (en este caso, A o B).

La hoja de cálculo `dimensions3.xlsx` es como la figura 14.6.

	A	B	
1	Tall row		
2		Wide column	
3			

Figura 14.6. La fila 1 y la columna B se han configurado con mayor altura y anchura.

La altura y la anchura predeterminada de las celdas varía entre versiones de Excel y `openpyxl`.

Combinar y separar celdas

Podemos combinar un grupo rectangular de celdas en una única celda con el método `merge_cells()` de la hoja. Escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > import openpyxl

> > > wb = openpyxl.Workbook()

> > > sheet = wb['Sheet']

> > > sheet.merge_cells('A1:D3') # Combina todas estas celdas.

> > > sheet['A1'] = 'Twelve cells merged together.'

> > > sheet.merge_cells('C5:D5') # Combina estas dos celdas.

> > > sheet['C5'] = 'Two merged cells.'

> > > wb.save('merged3.xlsx')
```

El argumento de `merge_cells()` es una simple cadena de texto de las

celdas superior izquierda e inferior derecha del área que hay que fusionar: ‘A1:D3’ combina 12 celdas en una sola. Para establecer el valor de estas celdas combinadas, basta con establecer el de la celda superior izquierda del grupo combinado.

Al ejecutar este código, merged.xlsx tendrá el aspecto de la figura 14.7.

	A	B	C	D	E
1	Twelve cells merged together.				
2					
3					
4			Two merged cells.		
5					
6					
7					

Figura 14.7. Celdas combinadas en una hoja de cálculo.

Para dividir celdas, llamamos al método `unmerge_cells()`:

```
> > > import openpyxl
> > > wb = openpyxl.load_workbook('merged3.xlsx')
> > > sheet = wb['Sheet']
> > > sheet.unmerge_cells('A1:D3') # Divide estas celdas.
> > > sheet.unmerge_cells('C5:D5')
> > > wb.save('unmerged3.xlsx')
```

Si guardas los cambios y echas un vistazo a la hoja de cálculo, verás que las celdas combinadas vuelven a ser celdas individuales.

Inmovilizar paneles

Para hojas de cálculo que son demasiado grandes como para ver todo el contenido a la vez, es útil inmovilizar algunas de las filas superiores o las columnas más a la izquierda de la pantalla. Los encabezados de filas o columnas “congelados”, por ejemplo, estarán siempre visibles para el usuario, aunque se desplace por la hoja de cálculo. Esto se conoce como inmovilizar paneles. En `openpyxl`,

cada objeto Worksheet tiene un atributo `freeze_panes` que se puede establecer en un objeto Cell o en una cadena de las coordenadas de la celda. Ten en cuenta que este atributo inmovilizará todas las filas por encima de esta celda y todas las columnas a su izquierda, pero no la fila y la columna de la celda de la llamada en sí. Para desinmovilizar todos los paneles, establece `freeze_panes` en `None` o `'A1'`. La tabla 14.3 muestra qué filas y columnas se inmovilizan para varios ejemplos de `freeze_panes`.

Tabla 14.3. Ejemplos de panel inmovilizado.

Ejemplos de configuración de `freeze_panes`

<code>sheet.freeze_panes = 'A2'</code>	Inmoviliza columnas
<code>sheet.freeze_panes = 'A2:B1'</code>	Inmoviliza filas
<code>sheet.freeze_panes = 'A2:B1:C1'</code>	Inmoviliza filas y columnas
<code>sheet.freeze_panes = 'A1'</code>	Inmoviliza la primera fila y la primera columna
<code>sheet.freeze_panes = None</code>	Desinmoviliza

Descarga otra copia de la hoja de cálculo `produceSales3.xlsx` e introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > import openpyxl

> > > wb = openpyxl.load_workbook('produceSales3.xlsx')

> > > sheet = wb.active

> > > sheet.freeze_panes = 'A2' # Inmoviliza las filas por encima de A2.

> > > wb.save('freezeExample3.xlsx')
```

Puedes ver el resultado en la figura 14.8.

	A	B	C	D	E	F
1	PRODUCE	COST PER POUND	POUNDS SOLD	TOTAL		
1591	Fava beans	2,69	0,7	1,88		
1592	Grapefruit	0,76	28,5	21,66		
1593	Green peppers	1,89	37	69,93		
1594	Watermelon	0,66	30,4	20,06		
1595	Celery	3,07	36,6	112,36		
1596	Strawberries	4,4	5,5	24,2		
1597	Green beans	2,52	40	100,8		

Sheet +

Listo

100 %

Figura 14.8. Fila 1 inmovilizada.

Como hemos establecido el atributo `freeze_panes` en 'A2', la fila 1 permanece visible, al margen de dónde se mueva el usuario en la hoja de cálculo.

Gráficos

El módulo `openpyxl` es compatible con gráficos de barras, de líneas, de dispersión y circulares, usando los datos de las celdas de la hoja. Para crear un gráfico, hay que hacer lo siguiente:

1. Crear un objeto `Reference` a partir de una selección rectangular de celdas.
2. Crear un objeto `Series` pasando el objeto `Reference`.
3. Crear un objeto `Chart`.
4. Adjuntar el objeto `Series` al objeto `Chart`.
5. Añadir el objeto `Chart` al objeto `Worksheet`, especificando opcionalmente qué celda debería ser la esquina superior izquierda del gráfico.

El objeto `Reference` requiere una explicación. Para crear objetos `Reference`, debemos llamar a la función `openpyxl.chart.Reference()` y pasarle cinco argumentos:

- El objeto `Worksheet` que contiene los datos del gráfico.
- Los enteros de fila y columna de la celda superior izquierda de la

selección rectangular de celdas que contienen los datos: el primer entero de la tupla es la fila y el segundo la columna. Observa que la primera fila es 1, no 0.

• Los enteros de fila y columna de la celda inferior derecha de la selección rectangular de celdas que contienen los datos: el primer entero de la tupla es la fila y el segundo la columna.

Pon este ejemplo en el intérprete de comandos interactivo para crear un gráfico de barras y añadirlo a la hoja de cálculo:

```
> > > import openpyxl

> > > wb = openpyxl.Workbook()

> > > sheet = wb.active

> > > for i in range(1, 11): # Crea datos en la columna A.
... sheet['A' + str(i)] = i * i
.

> > > ref_obj = openpyxl.chart.Reference(sheet, 1, 1, 1, 10)

> > > series_obj = openpyxl.chart.Series(ref_obj, title='First
series')

> > > chart_obj = openpyxl.chart.BarChart()

> > > chart_obj.title = 'My Chart'

> > > chart_obj.append(series_obj)

> > > sheet.add_chart(chart_obj, 'C5')

> > > wb.save('sampleChart3.xlsx')
```

Este código produce una hoja de cálculo como la de la figura 14.9.

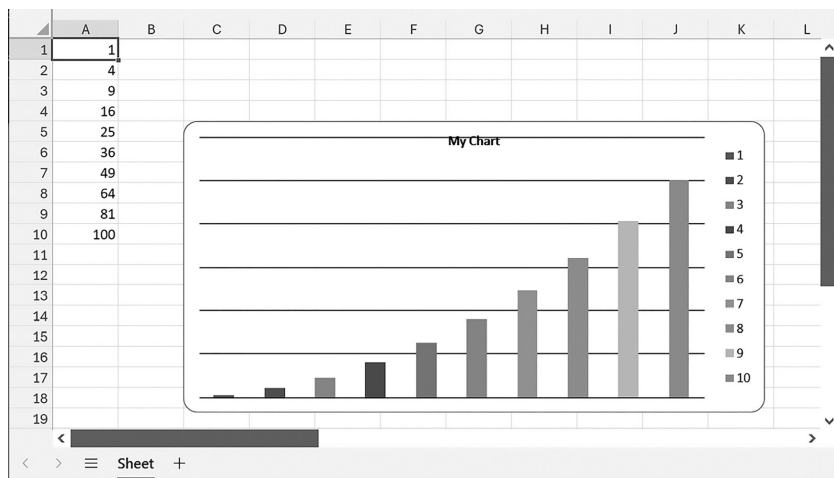


Figura 14.9. Una hoja de cálculo con un gráfico.

Hemos creado un gráfico de barras llamando a `openpyxl.chart.BarChart()`. También podemos crear gráficos de líneas, de dispersión o circulares llamando a `openpyxl.chart.LineChart()`, `openpyxl.chart.ScatterChart()` y `openpyxl.chart.PieChart()`, respectivamente.

Resumen

A menudo, lo más difícil de procesar información no es el procesamiento en sí, sino poner los datos en el formato adecuado para el programa. Pero, una vez cargada la hoja de cálculo de Excel en el programa de Python, se puede extraer y manipular datos mucho más rápido que a mano.

También se puede generar hojas de cálculo como salida de un programa. Así pues, si tus compañeros necesitan un archivo de texto o un PDF con miles de datos de contacto pasado a una hoja de cálculo, no tendrás que ir copiando y pegando todo en Excel. Equipado con el módulo `openpyxl` y un poco de conocimiento de programación, te parecerá pan comido procesar incluso las hojas de cálculo más grandes.

En el próximo capítulo, veremos cómo usar Python para interactuar con otro programa de hojas de cálculo: la popular aplicación en

línea Hojas de cálculo de Google.

Preguntas prácticas

Para las siguientes preguntas, imagina que tienes un objeto Workbook en la variable `wb`, un objeto Worksheet en `sheet` y un objeto Sheet en `sheet`.

- 1.¿Qué devuelve la función `openpyxl.load_workbook()`?
- 2.¿Qué contiene el atributo de libro `wb.sheetnames`?
- 3.¿Cómo recuperamos el objeto Worksheet de una hoja llamada 'Sheet1'?
- 4.¿Cómo recuperamos el objeto Worksheet para la hoja activa del libro?
- 5.¿Cómo recuperarías el valor de la celda C5?
- 6.¿Cómo establecerías el valor de la celda C5 en "Hello"?
- 7.¿Cómo recuperarías la fila y la columna de la celda como enteros?
- 8.¿Qué contienen los atributos de hoja `sheet.max_column` y `sheet.max_row` y cuál es el tipo de datos de estos atributos?
- 9.Si tuvieses que obtener el índice entero para la columna 'M', ¿a qué función deberías llamar?
- 10.Si necesitas obtener la cadena del nombre de la fila 14, ¿a qué función deberías llamar?
- 11.¿Cómo recuperamos una tupla con todos los objetos Cell de A1 a F1?
- 12.¿Cómo guardarías el libro con el nombre de archivo `example3.xlsx`?
- 13.¿Cómo se establece una fórmula en una celda?
- 14.Si se quiere recuperar el resultado de la fórmula de una celda en

vez de la fórmula en sí, ¿qué hay que hacer primero?

15. ¿Cómo establecerías la altura de la fila 5 en 100?

16. ¿Cómo ocultarías la columna C?

17. ¿Qué es un panel inmovilizado?

18. ¿A qué cinco funciones y métodos hay que llamar para crear un gráfico de barras?

Programas de práctica

Para practicar, escribe programas que realicen las siguientes tareas.

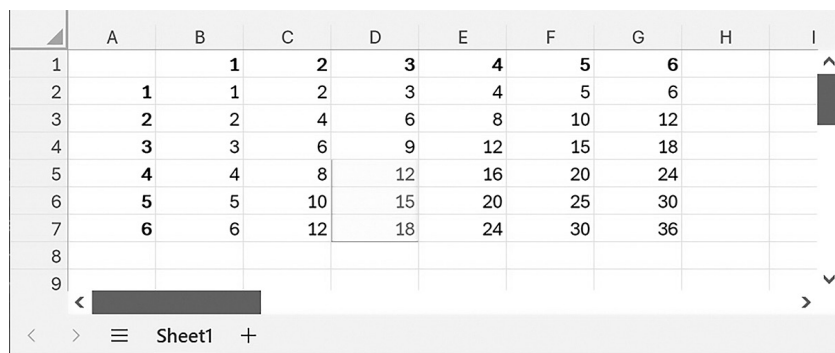
Tabla de multiplicar

Crea un programa `multiplicationTable.py` que coja un número N de la línea de comandos y cree una tabla de multiplicar $N \times N$ en una hoja de cálculo de Excel. Por ejemplo, al ejecutar el programa así:

`py multiplicationTable.py 6`

debería crear una hoja de cálculo como la que muestra la figura 14.10.

La fila 1 y la columna A deberían contener etiquetas en negrita.



	A	B	C	D	E	F	G	H	I
1		1	2	3	4	5	6		
2	1	1	2	3	4	5	6		
3	2	2	4	6	8	10	12		
4	3	3	6	9	12	15	18		
5	4	4	8	12	16	20	24		
6	5	5	10	15	20	25	30		
7	6	6	12	18	24	30	36		
8									
9									

Figura 14.10. Una tabla de multiplicar generada en una hoja de cálculo.

Crea un programa `blankRowInserter.py` que tome dos enteros y una cadena de nombre de archivo como argumentos de línea de comandos. Vamos a llamar al primer entero N y al segundo M . Empezando en la fila N , el programa debería insertar M filas en blanco en la hoja de cálculo. Al ejecutar el programa así:

```
python blankRowInserter.py 3 2 myProduce.xlsx
```

el “antes y después” de la hoja de cálculo debería ser como la figura 14.11.

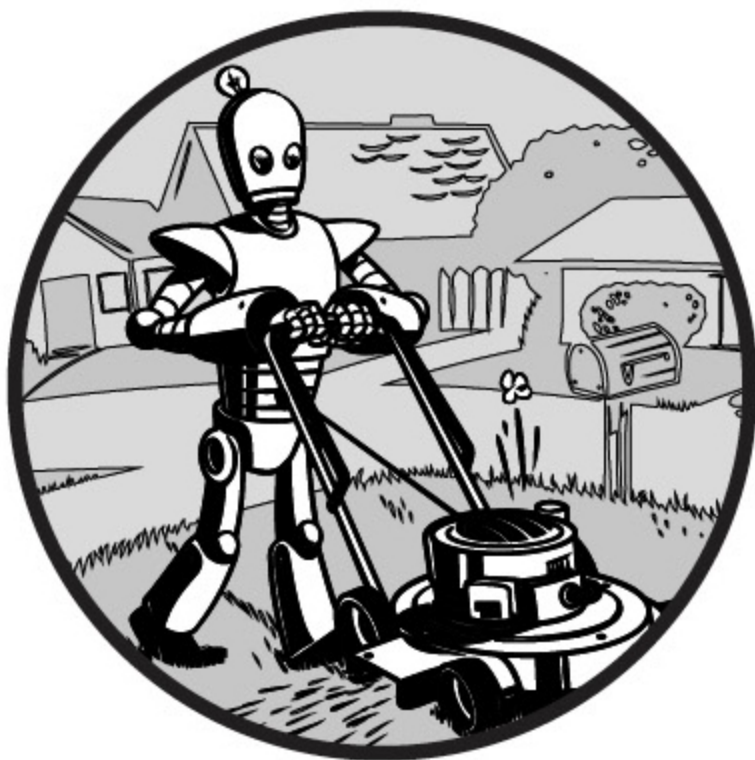
A1 Potatoes						
A	B	C	D	E	F	
1 Potatoes	Celery	Ginger	Yellow pepper	Green bean	Fava bean	
2 Okra	Okra	Corn	Garlic	Tomatoes	Yellow	
3 Fava bean	Spinach	Grapefruit	Grapes	Apricots	Papaya	
4 Watermelon	Cucumber	Ginger	Watermelon	Red onion	Butter	
5 Garlic	Apricots	Eggplant	Cherries	Strawberry	Apricot	
6 Parsnips	Okra	Cucumber	Apples	Grapes	Avocado	
7 Asparagus	Fava bean	Green bean	Grapefruit	Ginger	Butter	
8 Avocados	Watermelon	Eggplant	Grapes	Strawberry	Celery	

A1 Potatoes						
A	B	C	D	E	F	
1 Potatoes	Celery	Ginger	Yellow pepper	Green bean	Fava bean	
2 Okra	Okra	Corn	Garlic	Tomatoes	Yellow	
3						
4						
5 Fava bean	Spinach	Grapefruit	Grapes	Apricots	Papaya	
6 Watermelon	Cucumber	Ginger	Watermelon	Red onion	Butter	
7 Garlic	Apricots	Eggplant	Cherries	Strawberry	Apricot	
8 Parsnips	Okra	Cucumber	Apples	Grapes	Avocado	

Figura 14.11. Antes (izquierda) y después (derecha) de insertar las dos filas en blanco en la fila 3.

Puedes escribir este programa leyendo el contenido de la hoja de cálculo. A continuación, al escribir la nueva hoja, usa un bucle `for` para copiar las primeras N líneas. Para el resto de líneas, añade M al número de fila en la hoja de salida.

Hojas de cálculo de Google



Hojas de cálculo de Google, la aplicación de hojas de cálculo basada en web gratuita y disponible para cualquiera con una cuenta en Google o una dirección de Gmail, se ha convertido en un competidor de Excel muy útil y con muchas funciones. Hojas de cálculo de Google cuenta con su propia API, aunque puede ser difícil de aprender y usar. Este capítulo aborda la librería de terceros EZSheets, que ofrece una forma más sencilla de realizar acciones habituales, gestionando los detalles de la API para que tú no tengas que aprenderlos.

Instalar y configurar EZSheets

Puedes instalar EZSheets con la herramienta de línea de comandos `pip` siguiendo las instrucciones del apéndice A.

Para que tus *scripts* de Python puedan usar EZSheets para acceder a tus Hojas de cálculo de Google y editarlas, necesitas un archivo JSON de credenciales y dos archivos JSON de *tokens*. Crear las credenciales implica cinco pasos:

1. Crear un nuevo proyecto en Google Cloud.
2. Habilitar las API de Hojas de cálculo de Google y Google Drive para el proyecto.
3. Configurar la pantalla de consentimiento de OAuth.
4. Crear las credenciales.
5. Iniciar sesión con el archivo de credenciales.

Esto puede parecer mucho trabajo, pero solo hay que hacerlo una vez, y es gratis. Necesitarás una cuenta de Google/Gmail; te recomiendo encarecidamente que crees una nueva en lugar de emplear una que ya tengas para evitar que tu *script* de Python afecte a las hojas que haya en tu cuenta personal de Google. A lo largo de este capítulo, usaré cuenta de Google y dirección de Gmail para referirme a la cuenta que posee las hojas de cálculo a las que accede tu programa de Python.

Google puede cambiar ligeramente la disposición o redacción de la consola de su sitio web Google Cloud, pero los pasos anteriores deberían ser los mismos.

Crear un nuevo proyecto en Google Cloud

Primero, hay que crear un proyecto en Google Cloud. En tu navegador, ve a <https://console.cloud.google.com> e inicia sesión con tu cuenta de Google (usuario y contraseña). Llegarás a una página introductoria. En la parte superior de la página, haz clic en Seleccionar un proyecto. En la ventana emergente que aparecerá, haz clic en Proyecto nuevo. Esto debería llevarte a una página de proyecto nuevo.

Google Cloud generará un nombre de proyecto como My Project 23135, junto con un ID de proyecto aleatorio, como macro-nuance-362516. Estos valores no serán visibles para los usuarios de tus *scripts* de Python y puedes cambiar el nombre del proyecto por el que quieras, pero no así el ID. Yo uso el nombre que me genera el sitio web. Puedes dejar la ubicación como Sin organización. Las cuentas gratuitas de Google pueden tener hasta 12 proyectos, pero solo necesitamos uno para todos los *scripts* de Python que creemos. Haz clic en el botón azul Crear para crear el proyecto.

Habilitar las API de Hojas de cálculo y Drive

En la página <https://console.cloud.google.com>, haz clic en el botón Navegación de la esquina superior izquierda. (El icono tiene tres barras horizontales y se conoce como “menú hamburguesa”). Ve a APIs y servicios > Biblioteca para consultar la biblioteca de API. Verás API para Gmail, Google Maps, Google Cloud Storage y otros servicios de Google. Tenemos que permitir que el proyecto use las API de Hojas de cálculo de Google y Google Drive. EZSheets utiliza la API de Google Drive para cargar y descargar hojas de cálculo.

Baja hasta la API de Hojas de cálculo de Google y haz clic en ella, o escribe “**Hojas de cálculo de Google**” en la barra de búsqueda para localizarla. Esto debería llevarte a la página de la API. Haz clic en el botón azul Habilitar para permitir que tu proyecto de Google Cloud utilice esta API. Serás redirigido a la página APIs y servicios > APIs y servicios habilitados, donde encontrarás información sobre la frecuencia con la que tus *scripts* de Python están usando esta API. Repite este proceso para habilitar también la API de Google Drive.

A continuación, hay que configurar la pantalla de consentimiento de OAuth del proyecto.

Configurar la pantalla de consentimiento de OAuth

La pantalla de consentimiento de OAuth aparecerá al usuario la primera vez que ejecute `import ezsheets`. En el paso 1 de la pantalla, selecciona Externo y haz clic en el botón Crear. La siguiente página debería mostrar cómo queda la pantalla de consentimiento. Rellena el campo para el nombre (yo suelo usar

algo genérico, como “Python Google API Script”) y pon tu dirección de correo electrónico como información de contacto. Haz clic en el botón Guardar y continuar.

En el paso 2, define los ámbitos para tu proyecto o los recursos a los que puede acceder. Haz clic en el botón para añadir y quitar ámbitos y, en el panel que aparecerá, repasa la tabla y selecciona las casillas de verificación para los ámbitos .../auth/drive (la API de Google Drive) y .../auth/spreadsheets (thla API de Hojas de cálculo de Google). A continuación, haz clic en Actualizar y, después, en Guardar y continuar.

La página del paso 3 requiere añadir las direcciones de Gmail de las cuentas de Google que poseen las hojas de cálculo con las que interactuará el *script* de Python. A menos que pases por el proceso de aprobación de aplicaciones de Google, tus *scripts* están limitados a interactuar con las direcciones que añadidas en este paso. Haz clic en el botón + Agregar usuarios. En el panel que aparecerá, escribe la dirección de Gmail de tu cuenta de Google, haz clic en el botón azul y, después, en Guardar y continuar.

El paso 4 muestra un resumen de los pasos anteriores. Si toda la información está bien, haz clic en el botón Volver al panel. El siguiente paso es crear credenciales para tu proyecto.

Crear credenciales

Primero, tienes que crear un archivo de credenciales. EZSheets lo necesita para utilizar la API de Google, incluso para hojas de cálculo compartidas públicamente. En el menú Navegación, haz clic en APIs y servicios > Credenciales. A continuación, haz clic en el enlace Crear credenciales en la parte superior de la página. Se abrirá un submenú preguntando qué tipo de credenciales quieres crear: claves de API, ID de cliente de OAuth o cuenta de servicio. Haz clic en ID de cliente de OAuth.

En la siguiente página, selecciona App de escritorio en Tipo de aplicación y deja en Nombre el predeterminado Desktop client 1. Puedes cambiarlo si quieres, los usuarios de tu *script* de Python no lo verán. Haz clic en el botón Crear.

En la ventana emergente que se abrirá, haz clic en el botón de descarga para descargar el archivo JSON con las credenciales, que debería tener un nombre como `client_secret_2827922_35794-p2o9gfcub4htibfg2u207gcomco9nqm7.apps.googleusercontent.com.json`. Ponlo en la misma carpeta que tu *script* de Python. Para simplificar, puedes renombrar el archivo JSON como `credentials-sheets.json`. EZSheets busca `credentials-sheets.json` o cualquier archivo que coincida con el formato `client_secret_*.json`.

Iniciar sesión con el archivo de credenciales

Ejecuta el intérprete de comandos interactivo de Python desde la carpeta en la que está el archivo JSON de credenciales y, a continuación, ejecuta `import ezsheets`. EZSheets comprueba automáticamente el directorio de trabajo actual en busca de un archivo de JSON credenciales llamando a la función `ezsheets.init()`. Si lo encuentra, EZSheets abre el navegador web con la pantalla de consentimiento de OAuth para generar archivos de *tokens*. EZSheets también requiere estos archivos, llamados `token-drive.pickle` y `token-sheets.pickle`, además del de credenciales para acceder a las hojas de cálculo. Generar archivos de *tokens* es un paso de configuración único que no se repetirá la siguiente vez que ejecutemos `import ezsheets`.

Inicia sesión con tu cuenta de Google. Debe ser la misma dirección de correo que proporcionaste como usuario de prueba al configurar la pantalla de consentimiento de OAuth del proyecto en Google Cloud. Debería salirte un mensaje advirtiéndote que Google no ha verificado esta *app*, lo cual no es grave porque tú eres el creador de la *app*. Haz clic en Continuar. Deberías llegar a otra página que diga algo como “Python Google API Script quiere acceder a tu cuenta de Google” (con el nombre que pusieras al configurar la pantalla de consentimiento de OAuth). Haz clic en Continuar. Llegarás a una página web sencilla que dirá que la autenticación se ha completado. Ya puedes cerrar la ventana del navegador.

Una vez completado el proceso de autenticación para la API de Hojas de cálculo, debes repetir el proceso para la API de Drive en la ventana que se abrirá a continuación. Después de cerrar la segunda ventana, deberías ver los archivos `token-drive.pickle` y `token-`

sheets.pickle en la misma carpeta que el archivo JSON de credenciales. Trata estos archivos como contraseñas y no los compartas: pueden usarse para iniciar sesión y acceder a tus hojas de cálculo de Google.

Revocar el archivo de credenciales

Si, por accidente, has compartido los archivos de credenciales o *tokens* con alguien, no podrán cambiar la contraseña de tu cuenta de Google, pero tendrán acceso a tus hojas de cálculo. Puedes revocar estos archivos iniciando sesión en <https://console.developers.google.com>. Haz clic en Credenciales, en el menú lateral. A continuación, en la tabla de ID de cliente de OAuth 2.0, haz clic en el icono de la papelera junto al archivo de credenciales que has compartido sin querer. Una vez revocado, los archivos de credenciales y *tokens* quedan inservibles, así que puedes borrarlos. Tendrás que generar un nuevo archivo JSON de credenciales y nuevos archivos de *tokens*.

Objetos Spreadsheet

En Hojas de cálculo de Google, una hoja de cálculo puede contener varias hojas, y cada hoja contiene columnas y filas de celdas. Las celdas pueden contener datos como números, fechas o texto. También tienen propiedades, como la fuente, la altura y la anchura o el color de fondo. La figura 15.1 muestra una hoja de cálculo titulada Sweigart Books que contiene dos hojas, llamadas Books y Websites. Puedes ver esta hoja de cálculo en tu navegador visitando <https://autbor.com/examplegs>. La primera columna de cada hoja lleva la etiqueta A, y la primera fila, la etiqueta 1. (Esto no es como en las listas de Python, cuyo primer elemento está en el índice 0).

The screenshot shows a Google Spreadsheet interface. The title bar reads 'Sweigart Books (DO NOT DELETE)'. The spreadsheet has four columns: A (TITLE), B (AUTHOR), C (YEAR PUBLISHED), and D (URL). The data is as follows:

	A	B	C	D
1	TITLE	AUTHOR	YEAR PUBLISHED	URL
2	Automate the Boring Stuff with Python, 2nd Edition	Al Sweigart	2019	https://automatetheboringstuff.com/
3	The Recursive Book of Recursion	Al Sweigart	2022	https://inventwithpython.com/recursion/
4	Python Programming Exercises, Gently Explained	Al Sweigart	2022	https://inventwithpython.com/pythongently/
5	The Big Book of Small Python Projects	Al Sweigart	2021	https://inventwithpython.com/bigbookpython/
6	Beyond the Basic Stuff with Python	Al Sweigart	2020	https://inventwithpython.com/beyond/
7	Coding with Minecraft	Al Sweigart	2018	https://turtleappstore.com/book/
8	Cracking Codes with Python	Al Sweigart	2018	https://inventwithpython.com/cracking/
9	Invent Your Own Computer Games with Python, 4th	Al Sweigart	2016	https://inventwithpython.com/invent4thed/
10	Scratch 3 Programming Playground	Al Sweigart	2021	https://inventwithscratch.com/book3/
11	Making Games with Python & Pygame	Al Sweigart	2012	https://inventwithpython.com/pygame/

Figura 15.1. Una hoja de cálculo titulada Sweigart Books con dos hojas, Books y Websites.

Aunque la mayor parte de tu trabajo implicará modificar los objetos Sheet, también puedes modificar objetos Spreadsheet, como veremos en el siguiente apartado.

Crear, actualizar y hacer listas de hojas de cálculo

Podemos crear un nuevo objeto Spreadsheet a partir de una hoja de cálculo de Google ya existente, una hoja en blanco o una hoja de Excel cargada. Todas las hojas de cálculo de Google tienen un ID único que se encuentra en su URL, después de la parte `spreadsheets/d/` y antes de `/edit`. Por ejemplo, en la URL https://docs.google.com/spreadsheets/d/1TzOJxhNKr15_tzdZxTqtQ3EmDP6em_elnbtmZlcyu8vI/edit#gid=0/, el ID sería `1TzOJxhNKr15_tzdZxTqtQ3EmDP6em_elnbtmZlcyu8vI`.

Una hoja de cálculo de Google está representada como un objeto `ezsheets.Spreadsheet`, que tiene atributos `id`, `url` y `title`. Podemos crear una nueva hoja de cálculo en blanco con la función `Spreadsheet()`:

```
> > > import ezsheets
```

```
> > > ss = ezsheets.Spreadsheet()
```

```
> > > ss.title = 'Title of My New Spreadsheet'
```

```
> > > ss.title
```

```
'Title of My New Spreadsheet'
```

```
> > > ss.url
```

```
'https://docs.google.com/spreadsheets/d/1gxz-Qr2-RNtqi\_d7wWlsDlbtPLRQigcEXvCtdVwmH40/'
```

```
> > > ss.id
```

```
'1gxz-Qr2-RNtqi_d7wWlsDlbtPLRQigcEXvCtdVwmH40'
```

También podemos cargar una hoja de cálculo existente pasando su ID o URL, o una URL que redirija a su URL:

```
> > > import ezsheets
```

```
> > > ss1 = ezsheets.Spreadsheet('https://autbor.com/  
examples')
```

```
> > > ss2 = ezsheets.Spreadsheet('https://docs.google.com/  
spreadsheets/d/1TzOJxh
```

```
NKr15tzdZxTqtQ3EmDP6em_elnbtmZlcyu8vI/'))
```

```
> > > ss3 =
```

```
ezsheets.Spreadsheet('1TzOJxhNKr15tzdZxTqtQ3EmDP6em_elnbtmZlcyu8vI')
```

```
> > > ss1 = ss2 = ss3 # Son la misma hoja de cálculo.
```

True

Para cargar una hoja de Excel, OpenOffice, CSV o TSV existente en Hojas de cálculo de Google, pasa el nombre del archivo a `ezsheets.upload()`. Introduce lo siguiente en el intérprete de comandos interactivo, cambiando `my_spreadsheet.xlsx` por un archivo tuyo:

```
> > > import ezsheets
```

```
> > > ss = ezsheets.upload('my_spreadsheet.xlsx')
```

```
> > > ss.title
```

```
'my_spreadsheet'
```

Podemos hacer una lista con las hojas de cálculo de nuestra cuenta de Google llamando a la función `listSpreadsheets()`, que devuelve un diccionario cuyas claves son los ID de las hojas de cálculo y los valores los títulos de cada hoja. También incluye las hojas eliminadas que estén en la carpeta Papelera. Prueba a introducir lo siguiente en el intérprete de comandos interactivo después de cargar una hoja de cálculo:

```
> > > ezsheets.listSpreadsheets()
```

```
{'1J-Jx6Ne2K_vqI9J2SO-TAXOFbxx_9tUjwnkPC22LjeU': 'Education Data'}
```

Una vez que tenemos un objeto Spreadsheet, podemos usar sus métodos y atributos para manipular la hoja de cálculo en línea alojada en Hojas de cálculo de Google Sheets.

Acceder a los atributos de una hoja de cálculo

Aunque los datos se encuentran en las hojas individuales de una hoja de cálculo, el objeto Spreadsheet tiene los siguientes atributos para manipular la propia hoja: `title`, `id`, `url`, `sheetTitles` y `sheets`.

Vamos a analizar la hoja de <https://autbor.com/examples>. Tu cuenta de Google tiene permiso para verla, pero no para modificarla; aun así, puedes copiarla en una hoja recién creada en tu propia cuenta:

```
> > > import ezsheets
```

```
> > > example_ss = ezsheets.Spreadsheet('https://  
autbor.com/examples)
```

```
> > > ss = ezsheets.Spreadsheet()
```



```
> > > example_ss.sheets[0].copyTo(ss)
```

```
> > > ss.sheets[0].delete() # Elimina la hoja Sheet1.
```

```
> > > ss.url
```

```
https://docs.google.com/spreadsheets/  
d/15gjrBgTmUzItRt9KUcL4JajLaQU70xanstB1dXKoSlM/
```

La hoja recién copiada indicará en el nombre que es una copia de Books, ya que Books era el nombre de la hoja original. Continúa el ejemplo del intérprete de comandos interactivo con el siguiente código:

```
> > > ss.title # El nombre de la hoja
```

```
‘Untitled spreadsheet’
```

```
> > > ss.title = ‘Sweigart Books’ # Cambia el nombre.
```

```
> > > ss.id # El ID único (atributo de solo lectura)
```

```
‘15gjrBgTmUzItRt9KUcL4JajLaQU70xanstB1dXKoSlM’
```

```
> > > ss.url # La URL original URL (atributo de solo lectura)
```

```
https://docs.google.com/spreadsheets/  
d/15gjrBgTmUzItRt9KUcL4JajLaQU70xanstB1dXKoSlM/
```

```
> > > ss.sheetTitles # Los nombres de todos los objetos Sheet
```

```
(‘Copy of Books’,)
```

```
> > > ss.sheets # Los objetos Sheet de esta hoja, en orden
```

```
(<Sheet sheetId=1464919459, title=‘Copy of Books’,  
rowCount=1000, columnCount=26>,)
```

```
> > > ss.sheets[0] # el primer objeto Sheet de esta hoja
```

```
<Sheet sheetId=1464919459, title=‘Copy of Books’,  
rowCount=1000, columnCount=26>
```

> > > **ss['Copy of Books']** # También se puede acceder a una hoja por el nombre.

```
<Sheet sheetId=1464919459, title='Copy of Books',  
rowCount=1000, columnCount=26>
```

> > > **ss.Sheet('New blank sheet')** # Crea una hoja nueva.

```
<Sheet sheetId=1759616008, title='New blank sheet',  
rowCount=1000, columnCount=26>
```

> > > **ss.sheets[1].delete()** # Borra el segundo objeto Sheet de esta hoja.

Si alguien cambia la hoja de cálculo en su navegador, nuestro *script* puede actualizar el objeto Spreadsheet para que coincida con los datos en línea llamando al método `refresh()`:

> > > **ss.refresh()**

Esto actualizará no solo los atributos del objeto Spreadsheet, sino también los datos de los objetos Sheet que contiene. Verás los cambios realizados en el objeto Spreadsheet aplicados a la hoja de cálculo en línea en tiempo real.

Descargar y actualizar hojas de cálculo

Podemos descargar hojas de cálculo de Google en varios formatos: Excel, OpenOffice, CSV, TSV y PDF. También podemos descargarla como archivo ZIP con archivos HTML de los datos de la hoja. EZSheets contiene funciones para cada una de estas opciones:

> > > **import ezsheets**

> > > **ss = ezsheets.Spreadsheet('https://autbor.com/examples')**

> > > **ss.title**

'Sweigart Books (DO NOT DELETE)'

> > > **ss.downloadAsExcel()** # Descarga la hoja de cálculo como

archivo de Excel.

`'Sweigart_Books.xlsx'`

`> > > ss.downloadAsODS()` # Descarga la hoja de cálculo como archivo de OpenOffice.

`'Sweigart_Books.ods'`

`> > > ss.downloadAsCSV()` # Descarga la hoja de cálculo como archivo CSV.

`'Sweigart_Books.csv'`

`> > > ss.downloadAsTSV()` # Descarga solo la primera hoja como archivo TSV.

`'Sweigart_Books.tsv'`

`> > > ss.downloadAsPDF()` # Descarga la hoja de cálculo como PDF.

`'Sweigart_Books.pdf'`

`> > > ss.downloadAsHTML()` # Descarga la hoja de cálculo como archivo ZIP con archivos HTML.

`'Sweigart_Books.zip'`

Ten en cuenta que los archivos en formato CSV o TSV solo pueden contener una hoja, por tanto, si descargas una hoja de cálculo de Google en alguno de esos formatos, solo te saldrá la primera hoja. Para descargar otras, tendrás que reordenar los objetos Sheet antes de la descarga.

Todas las funciones de descarga devuelven una cadena del nombre del archivo descargado. También podemos especificar un nombre de archivo para la hoja pasándoselo a la función de descarga:

`> > > ss.downloadAsExcel('a_different_filename.xlsx')`

`'a_different_filename.xlsx'`

La función devuelve el nombre del archivo local.

Borrar hojas de cálculo

Para eliminar una hoja de cálculo, llama al método delete():

```
> > > import ezsheets
```

```
> > > ss = ezsheets.Spreadsheet() # Crea la hoja de cálculo.
```

```
> > > ezsheets.listSpreadsheets() # Confirma que hemos creado  
una hoja de cálculo.
```

```
{'1aCw2NNJSZblDbhygVv77kPsL3djmgV5zJZllSOZ_mRk': 'Delete  
me'}
```

```
> > > ss.delete() # Borra la hoja de cálculo.
```

```
> > > ezsheets.listSpreadsheets() # Las hojas de la carpeta  
Papelera siguen apareciendo en la lista.
```

```
{'1aCw2NNJSZblDbhygVv77kPsL3djmgV5zJZllSOZ_mRk': 'Delete  
me'}
```

El método delete() moverá nuestra hoja de cálculo a la Papelera de Google Drive. Puedes ver el contenido de esta carpeta en <https://drive.google.com/drive/trash>. Observa que las hojas que están en la carpeta Papelera siguen apareciendo en el diccionario que devuelve listSpreadsheets(). Para eliminar una hoja de cálculo de manera permanente, hay que pasar True para el argumento permanent:

```
> > > ss.delete(permanent = True)
```

```
> > > ezsheets.listSpreadsheets()
```

```
{}
```

En general, eliminar permanentemente una hoja de cálculo con *scripts* automatizados no es buena idea, porque es imposible recuperar una hoja que haya borrado por accidente un fallo en el script. De todas formas, las cuentas gratuitas de Google Drive disponen de gigabytes de almacenamiento, así que lo más probable

es que no tengas que preocuparte por liberar espacio.

Objetos Sheet

Un objeto Spreadsheet tendrá uno o varios objetos Sheet. Estos objetos representan las filas y columnas de datos de cada hoja. Podemos acceder a las hojas con corchetes como operador y un índice entero.

El atributo `sheets` del objeto Spreadsheet contiene una tupla de objetos Sheet en el orden en el que aparecen en la hoja de cálculo. Para acceder al objeto Sheet en una hoja de cálculo, escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > import ezsheets
```

```
> > > ss = ezsheets.Spreadsheet() # Empieza con una hoja  
llamada Sheet1.
```

```
> > > sheet2 = ss.Sheet('Spam')
```

```
> > > sheet3 = ss.Sheet('Eggs')
```

```
> > > ss.sheets # Los objetos Sheet de esta hoja de cálculo, en  
orden
```

```
( <Sheet sheetId = 0, title = 'Sheet1', rowCount = 1000,  
columnCount = 26 >, <Sheet sheetId = 284204004,
```

```
title = 'Spam', rowCount = 1000, columnCount = 26 >, <Sheet  
sheetId = 1920032872, title = 'Eggs',
```

```
rowCount = 1000, columnCount = 26 > )
```

```
> > > ss.sheets[0] # Obtiene el primer objeto Sheet de esta hoja  
de cálculo
```

```
<Sheet sheetId = 0, title = 'Sheet1', rowCount = 1000,  
columnCount = 26 >
```

El atributo `sheetTitles` del objeto Spreadsheet contiene una tupla con los nombres de todas las hojas. Para ver un ejemplo, introduce

esto en el intérprete de comandos interactivo:

```
> > > ss.sheetTitles # Los títulos de todos los objetos Sheet de esta hoja de cálculo
```

```
('Sheet1', 'Spam', 'Eggs')
```

Una vez que tenemos un objeto Sheet, podemos leer y escribir datos en él con los métodos del objeto que veremos en el próximo apartado.

Leer y escribir datos

Igual que en Excel, las Hojas de cálculo de Google tienen filas y columnas de celdas que contienen datos. Podemos usar [para leer y escribir datos en estas celdas. Por ejemplo, para crear una nueva hoja de cálculo y añadirle datos, escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > import ezsheets
```

```
> > > ss = ezsheets.Spreadsheet()
```

```
> > > ss.title = 'My Spreadsheet'
```

```
> > > sheet = ss.sheets[0] # Obtiene la primera hoja de esta hoja de cálculo.
```

```
> > > sheet.title
```

```
'Sheet1'
```

```
> > > sheet['A1'] = 'Name' # Establece el valor de la celda A1.
```

```
> > > sheet['B1'] = 'Age'
```

```
> > > sheet['C1'] = 'Favorite Movie'
```

```
> > > sheet['A1'] # Lee el valor en la celda A1.
```

```
'Name'
```

> > > **sheet['A2']** # Las celdas vacías devuelven una cadena en blanco.

“

> > > **sheet[2, 1]** # Columna 2, fila 1 es lo mismo que B1.

‘Age’

> > > **sheet['A2']** = ‘Alice’

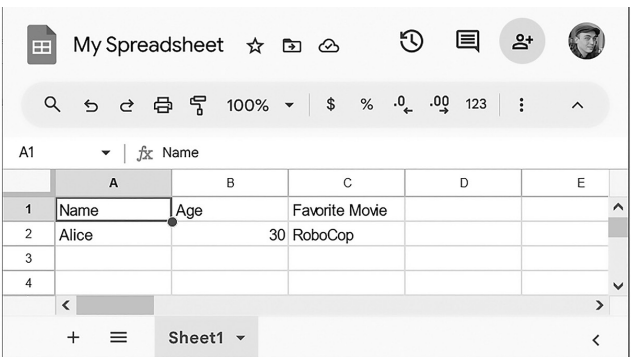
> > > **sheet['B2']** = 30

> > > **sheet['C2']** = ‘RoboCop’

> > > **sheet['B2']** # Observa que todos los datos se devuelven como cadenas.

‘30’

Estas instrucciones deberían producir una hoja de cálculo de Google como la de la figura 15.2.



	A	B	C	D	E
1	Name	Age	Favorite Movie		
2	Alice	30	RoboCop		
3					
4					

Figura 15.2. La hoja de cálculo creada con las instrucciones del ejemplo.

Todos los datos del objeto Sheet se cargan cuando se carga por primera vez el objeto Spreadsheet, de modo que los datos se leen al instante. Sin embargo, escribir valores en la hoja de cálculo en línea requiere una conexión de red y puede llevar aproximadamente un segundo. Si tienes miles de celdas que actualizar, hacerlo de una en

una puede ser muy lento. En lugar de eso, en los próximos apartados veremos cómo actualizar filas y columnas enteras de una vez.

Direcciones de filas y columnas

Las direcciones de filas y columnas funcionan en Hojas de cálculo de Google igual que en Excel. La única diferencia es que, a diferencia de los índices de lista de Python, que se empiezan en 0, las hojas de cálculo empiezan sus filas y columnas en 1: la primera columna o fila tiene el índice 1, no 0. Podemos cambiar el estilo de dirección de cadena 'A2' al estilo de tupla (column, row) (y viceversa) con la función `convertAddress()`. Las funciones `getColumnLetterOf()` y `getColumnNumberOf()` también convertirán una dirección de columna entre letras y números. Por ejemplo, introduce esto en el intérprete de comandos interactivo:

```
> > > import ezsheets
```

```
> > > ezsheets.convertAddress('A2') # Convierte direcciones...
```

```
(1, 2)
```

```
> > > ezsheets.convertAddress(1, 2) # ...y las convierte también al revés.
```

```
'A2'
```

```
> > > ezsheets.getColumnLetterOf(2)
```

```
'B'
```

```
> > > ezsheets.getColumnNumberOf('B')
```

```
2
```

```
> > > ezsheets.getColumnLetterOf(999)
```

```
'ALK'
```

```
> > > ezsheets.getColumnNumberOf('ZZZ')
```



```
> > > ss = ezsheets.upload('produceSales3.xlsx')
```

```
> > > sheet = ss.sheets[0]
```

```
> > > sheet.getRow(1) # La primera fila es 1, no 0.
```

```
['PRODUCE', 'COST PER POUND', 'POUNDS SOLD', 'TOTAL', '', '']
```

```
> > > sheet.getRow(2)
```

```
['Potatoes', '0.86', '21.6', '18.58', '', '']
```

```
> > > sheet.getColumn(1)
```

```
['PRODUCE', 'Potatoes', 'Okra', 'Fava beans', 'Watermelon', 'Garlic',
```

--fragmento omitido--

```
> > > sheet.getColumn('A') # El mismo resultado que  
getColumn(1)
```

```
['PRODUCE', 'Potatoes', 'Okra', 'Fava beans', 'Watermelon', 'Garlic',
```

--fragmento omitido--

```
> > > sheet.getRow(3)
```

```
['Okra', '2.26', '38.6', '87.24', '', '']
```

```
> > > sheet.updateRow(3, ['Pumpkin', '11.50', '20', '230'])
```

```
> > > sheet.getRow(3)
```

```
['Pumpkin', '11.50', '20', '230', '', '']
```

```
> > > columnOne = sheet.getColumn(1)
```

```
> > > for i, value in enumerate(columnOne):
```

```
... # Hace que la lista de Python contenga cadenas en mayúscula:
```

```
... columnOne[i] = value.upper()
```

.
> > > **sheet.updateColumn(1, columnOne)** # Actualiza toda la columna con una solicitud.

Las funciones `getRow()` y `getColumn()` recuperan los datos de todas las celdas de una fila o columna concreta como lista de valores. Las celdas en blanco se convertirán en cadenas en blanco en esa lista. Podemos pasar `getColumn()` a un número de columna o una letra para decirle que recupere los datos de una columna específica. El ejemplo anterior muestra que `getColumn(1)` y `getColumn('A')` devuelven la misma lista.

Las funciones `updateRow()` y `updateColumn()` sobrescribirán los datos de la fila o columna, respectivamente, con la lista de valores pasada a la función. En este ejemplo, la tercera fila contenía inicialmente información sobre okra, pero la llamada a `updateRow()` la sustituye por datos de las calabazas. Vuelve a llamar a `sheet.getRow(3)` para ver los nuevos valores en la tercera fila.

Actualizar las filas de una en una es lento si hay que actualizar muchas. Obtener una columna o fila como lista y después actualizar toda la columna o fila con la lista es mucho más rápido porque se puede hacer todos los cambios en una solicitud a los servicios de Google Cloud.

Para obtener todas las filas de una vez, llama al método `getRows()` para que devuelva una lista de listas. Las listas dentro de la lista exterior representan una sola fila de la hoja. Podemos modificar los valores de esta estructura para cambiar el nombre del producto, las libras vendidas y el coste total de algunas filas. Después, podemos pasar la lista al método `updateRows()` poniendo lo siguiente en el intérprete de comandos interactivo:

> > > **rows = sheet.getRows()** # Obtiene todas las filas de la hoja.

> > > **rows[0]** # Examina los valores de la primera fila.

['PRODUCE', 'COST PER POUND', 'POUNDS SOLD', 'TOTAL', ',', ',']

```
> > > rows[1]
```

```
['POTATOES', '0.86', '21.6', '18.58', ',', '']
```

```
> > > rows[1][0] = 'PUMPKIN' # Cambia el nombre del  
producto.
```

```
> > > rows[1]
```

```
['PUMPKIN', '0.86', '21.6', '18.58', ',', '']
```

```
> > > rows[10]
```

```
['OKRA', '2.26', '40', '90.4', ',', '']
```

```
> > > rows[10][2] = '400' # Cambia el peso vendido.
```

```
> > > rows[10][3] = '904' # Cambia el total.
```

```
> > > rows[10]
```

```
['OKRA', '2.26', '400', '904', ',', '']
```

```
> > > sheet.updateRows(rows) # Actualiza la hoja de cálculo en  
línea con los cambios.
```

Podemos actualizar toda la hoja con una solicitud pasando a `updateRows()` la lista de listas devuelta por `getRows()`, modificada con los cambios realizados en las filas de la 1 a la 10.

Fíjate en que las hojas de Hojas de cálculo de Google tienen cadenas en blanco al final. Esto se debe a que la hoja cargada tiene un número de columnas de 6, pero solo tenemos 4 columnas con datos. Podemos leer el número de filas y columnas de una hoja con los atributos `rowCount` y `columnCount`. Después, estableciendo estos valores, podemos cambiar el tamaño de la hoja:

```
> > > sheet.rowCount # El número de filas de la hoja
```

```
23758
```

```
> > > sheet.columnCount # El número de columnas de la hoja
```

> > > **sheet.columnCount = 4** # Cambia el número de columnas a 4.

> > > **sheet.columnCount** # Ahora el número de columnas de la hoja es 4.

Estas instrucciones deberían borrar las columnas quinta y sexta de la hoja produceSales3.xlsx, como muestra la figura 15.3.

Según la documentación de Google, las hojas de Hojas de cálculo de Google pueden contener hasta 10 millones de celdas. Sin embargo, es mejor hacer las hojas del tamaño estrictamente necesario para reducir el tiempo de actualización de los datos.

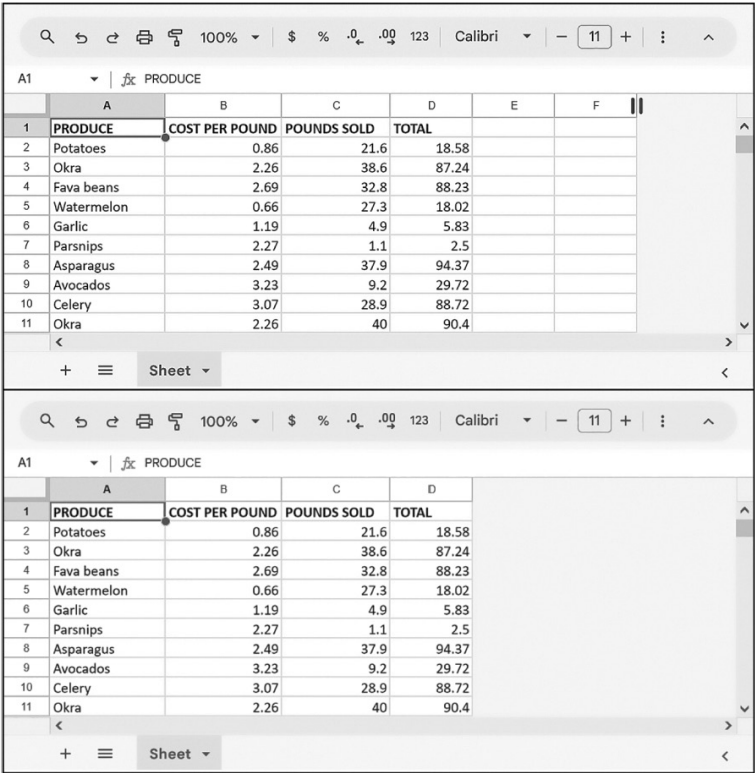


Figura 15.3. La hoja antes (arriba) y después (abajo) de cambiar el número de columnas a cuatro.

Crear, mover y eliminar hojas

Todas las Hojas de cálculo de Google empiezan con una sola hoja llamada Hoja1 (o Sheet1). Podemos añadir hojas adicionales al final de la lista de hojas con el método `Sheet()`, que acepta una cadena opcional que usar como nombre de la hoja nueva. Un segundo argumento opcional puede especificar el índice entero de la nueva hoja.

Para crear una hoja de cálculo y añadirle hojas, escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > import ezsheets
```

```
> > > ss = ezsheets.Spreadsheet()
```

```
> > > ss.title = 'Multiple Sheets'
```

```
> > > ss.sheetTitles
```

```
('Sheet1',)
```

```
> > > ss.Sheet('Spam') # Crea una hoja nueva al final de la lista de hojas.
```

```
<Sheet sheetId=2032744541, title='Spam', rowCount=1000, columnCount=26>
```

```
> > > ss.Sheet('Eggs') # Crea otra hoja nueva.
```

```
<Sheet sheetId=417452987, title='Eggs', rowCount=1000, columnCount=26>
```

```
> > > ss.sheetTitles
```

```
('Sheet1', 'Spam', 'Eggs')
```

```
> > > ss.Sheet('Bacon', 0) # Crea una hoja en el índice 0 de la lista de hojas.
```

```
<Sheet sheetId=814694991, title='Bacon', rowCount=1000, columnCount=26>
```

> > > **ss.sheetTitles**

('Bacon', 'Sheet1', 'Spam', 'Eggs')

Estas instrucciones añaden tres hojas nuevas a la hoja de cálculo: Bacon, Spam y Eggs (además de la predeterminada Hoja1). Las hojas de una hoja de cálculo están ordenadas y las hojas nuevas se van añadiendo al final de la lista, a menos que pasemos un segundo argumento a Sheet() especificando el índice de la hoja. Aquí, hemos creado la hoja Bacon en el índice 0, lo que la coloca como primera hoja, desplazando las otras tres una posición. Esto es similar al comportamiento del método de lista insert().

Puedes ver las nuevas hojas en las pestañas de abajo, como muestra la figura 15.4.

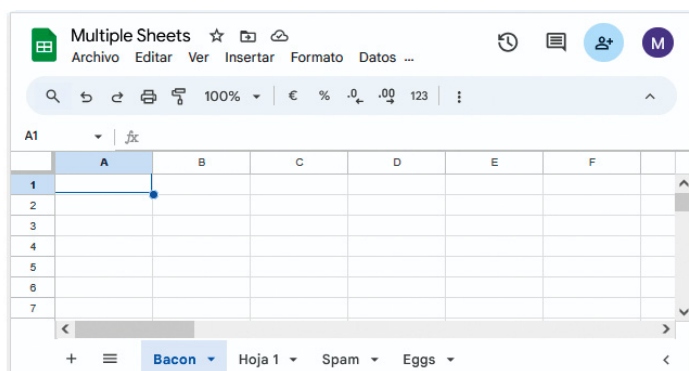


Figura 15.4. La hoja Multiple Sheets después de añadir las hojas Spam, Eggs y Bacon

Podemos cambiar el orden de una hoja con su atributo index y, después, asignar un nuevo índice a este atributo para reordenar la hoja:

> > > **ss.sheetTitles**

('Bacon', 'Sheet1', 'Spam', 'Eggs')

```
> > > ss.sheets[0].index
```

```
0
```

```
> > > ss.sheets[0].index = 2 # Mueve la hoja del índice 0 al  
índice 2.
```

```
> > > ss.sheetTitles
```

```
('Sheet1', 'Spam', 'Bacon', 'Eggs')
```

```
> > > ss.sheets[2].index = 0 # Mueve la hoja del índice 2 al  
índice 0.
```

```
> > > ss.sheetTitles
```

```
('Bacon', 'Sheet1', 'Spam', 'Eggs')
```

El método `delete()` del objeto `Sheet` eliminará la hoja de la hoja de cálculo. Si quieres conservar la hoja, pero borrar los datos que contiene, llama al método `clear()` para limpiar todas las celdas y dejar la hoja en blanco. Escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > ss.sheetTitles
```

```
('Bacon', 'Sheet1', 'Spam', 'Eggs')
```

```
> > > ss.sheets[0].delete() # Borra la hoja en el índice 0: la hoja  
"Bacon".
```

```
> > > ss.sheetTitles
```

```
('Sheet1', 'Spam', 'Eggs')
```

```
> > > ss['Spam'].delete() # Borra la hoja "Spam".
```

```
> > > ss.sheetTitles
```

```
('Sheet1', 'Eggs')
```

```
> > > sheet = ss['Eggs'] # Asigna una variable a la hoja "Eggs".
```



```
> > > sheet.delete() # Elimina la hoja “Eggs”.
```

```
> > > ss.sheetTitles
```

```
(‘Sheet1’,)
```

```
> > > ss.sheets[0].clear() # Borra todos los datos de “Hoja1”.
```

```
> > > ss.sheetTitles # La “Hoja1” está vacía pero sigue  
existiendo.
```

```
(‘Sheet1’,)
```

La eliminación es permanente; no hay manera de recuperar los datos. No obstante, podemos hacer una copia de seguridad en otra hoja de cálculo con el método `copyTo()`, que veremos en el siguiente apartado.

Copiar hojas

Todo objeto `Spreadsheet` tiene una lista ordenada de los objetos `Sheet` que contiene y podemos usar esa lista para reordenar las hojas (como acabamos de ver) o para copiarlas en otra hoja de cálculo. Para copiar un objeto `Sheet` en otro objeto `Spreadsheet`, hay que llamar al método `copyTo()` y pasarle el objeto `Spreadsheet` de destino como argumento. Para crear dos hojas de cálculo y copiar los datos de la primera en la segunda, introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > import ezsheets
```

```
> > > ss1 = ezsheets.Spreadsheet()
```

```
> > > ss1.title = ‘First Spreadsheet’
```

```
> > > ss1.sheets[0].title = ‘Spam’ # ss1 tendrá una hoja  
llamada Spam.
```

```
> > > ss2 = ezsheets.Spreadsheet()
```

```
> > > ss2.title = ‘Second Spreadsheet’
```

```
> > > ss2.sheets[0].title = 'Eggs' # ss2 tendrá una hoja llamada Eggs.
```

```
> > > ss1[0]
```

```
<Sheet sheetId=0, title='Spam', rowCount=1000,  
columnCount=26>
```

```
> > > ss1[0].updateRow(1, ['Some', 'data', 'in', 'the', 'first',  
'row'])
```

```
> > > ss1[0].copyTo(ss2) # Copia la Hoja1 de ss1 en la hoja de  
cálculo ss2.
```

```
> > > ss2.sheetTitles # ss2 contiene ahora una copia de la Hoja1  
de ss1.
```

```
('Eggs', 'Copy of Spam')
```

Las hojas copiadas aparecerán con el prefijo Copy of al final de la lista de hojas de la hoja de cálculo de destino. Si quieres, puedes cambiar su atributo index para reordenarlas en la nueva hoja de cálculo.

Formularios de Google

Tu cuenta de Google también te da acceso a Formularios de Google en <https://forms.google.com/>. Puedes crear encuestas, registros para eventos o formularios de comentarios con Formularios de Google y recibir las respuestas que envíen los usuarios en una hoja de cálculo de Hojas de cálculo de Google. Con EZSheets, tus programas de Python pueden acceder a estos datos de la hoja.

En el capítulo 19, veremos cómo diseñar programas de Python que se ejecuten regularmente a las horas especificadas. Podrías escribir un programa que compruebe con regularidad una hoja de cálculo de Formularios de Google para ver las respuestas y detectar cualquier nueva entrada que no hubiera antes. Después, con la información del capítulo 20, puedes hacer que el programa te envíe un mensaje para obtener notificaciones en tiempo real cuando se rellena un formulario.

Como ves, Python es célebre como lenguaje “pegamento” para unir distintos sistemas de software existente, permitiéndonos crear un proceso automatizado más potente que la suma de las partes.

Proyecto 11: Estafa de criptomonedas con blockchain falso

En este proyecto, vamos a utilizar Hojas de cálculo de Google como *blockchain* para hacer un seguimiento de las transacciones de Boringcoin, una estafa de criptomonedas que estoy promocionando. (Resulta que a los inversores y clientes les da igual si tu producto de *blockchain* usa una estructura de datos *blockchain* real; te van a dar el dinero de todos modos).

La URL <https://autbor.com> /boringcoin redirige a la URL de Hojas de cálculo de Google para el *blockchain* de Boringcoin. La hoja de cálculo tiene tres columnas: el emisor de la transacción, el receptor de la transacción y la cantidad de la transacción. La cantidad se resta al emisor y se suma al receptor. Si el emisor es ‘PRE-MINE’, ese dinero se ha creado de la nada y se ha añadido a la cuenta del receptor. La figura 15.5 muestra esta hoja de cálculo de Google Sheet.

La primera transacción tiene ‘PRE-MINE’ como emisor y ‘Al Sweigart’ como receptor y la cantidad es modesta, 1000000000. La cuenta ‘Al Sweigart’ transfiere a continuación 19116 Boringcoins a ‘Miles Bron’, que a su vez transfiere 118 Boringcoins a ‘not_a_scammer’. La cuarta transacción transfiere 16273 Boringcoins de ‘Al Sweigart’ a ‘some hacker’. (Yo no autoricé esta transacción y desde entonces ya no uso python12345 como contraseña en mi cuenta de Google).

	A	B	C
1	PRE-MINE	AI Sweigart	1000000000
2	AI Sweigart	Miles Bron	19116
3	Miles Bron	not_a_scammer	118
4	AI Sweigart	some hacker	16273
5	AI Sweigart	not_a_scammer	28356
6	some hacker	Tech Bro	52
7	some hacker	AI Sweigart	17
8	some hacker	not_a_scammer	40
9	Claire Debella	Credulous Journalist	57
10	AI Sweigart	Birdie Jay	30631
11	Birdie Jay	Miles Bron	125

Figura 15.5. El *blockchain* falso de Boringcoin, almacenado en una hoja de cálculo de Google.

Vamos a escribir dos programas. Primero, `auditBoringcoin.py` examinará todas las transacciones y generará un diccionario de todas las cuentas y su saldo actual. Después, `addBoringcoinTransaction.py` añadirá una fila al final de la hoja de cálculo de Google para una nueva transacción. Estos programas de *blockchain* son solo por diversión, no reales (aunque hay proyectos “reales” de *blockchain* como los NFT y “web3” que son igual de fantasiosos).

Paso 1: auditar el *blockchain* falso

Necesitamos escribir un programa que examine todo el “*blockchain*” y determine el saldo actual de todas las cuentas. Usaremos un diccionario para guardar estos datos, donde las claves serán las cadenas del nombre de cuenta y los valores los enteros de la cantidad de Boringcoins que tienen. También queremos que el programa muestre el total de Boringcoins que hay en la red de criptomonedas. Empezaremos importando `EZSheets` y creando el diccionario:

```
import ezsheets
```

```
ss = ezsheets.Spreadsheet('https://autbor.com/boringcoin')
```

accounts = {} # Las claves son nombres y los valores son cantidades.

A continuación, pasaremos en bucle por cada fila de la hoja de cálculo, identificando al emisor, al receptor y la cantidad. Ten en cuenta que Hojas de cálculo de Google devuelve siempre los datos como cadenas, así que habrá que convertirlos a enteros para poder hacer cálculos con el valor de amount:

Cada cadena es una transacción. Pasa en bucle por cada una:

```
for row in ss.sheets[0].getRows():
```

```
    sender, recipient, amount = row[0], row[1], int(row[2])
```

Si el emisor es la cuenta especial 'PRE-MINE', se trata de una fuente inagotable de dinero para otras cuentas. Las mejores estafas de criptomonedas usan monedas extraídas antes de salir públicamente, y la nuestra no es una excepción. Añade la cantidad a la cuenta del receptor en el diccionario accounts. El método.setdefault() establece el valor de la cuenta en 0 si no existe ya en el diccionario:

```
    if sender == 'PRE-MINE':
```

```
        # El emisor 'PRE-MINE' se saca el dinero de la manga.
```

```
        accounts.setdefault(recipient, 0)
```

```
        accounts[recipient] += amount
```

Si existe, deberíamos restar la cantidad del emisor y sumarla al receptor:

```
    else:
```

```
        # Mueve fondos del emisor al receptor.
```

```
        accounts.setdefault(sender, 0)
```

```
        accounts.setdefault(recipient, 0)
```

```
        accounts[sender] -= amount
```

```
accounts[recipient] += amount
```

Cuando termine el bucle, podremos ver los saldos actuales imprimiendo el diccionario `accounts`.

```
print(accounts)
```

Como parte de la auditoría, vamos también a pasar por este diccionario y sumar los totales del saldo de todos para ver cuántas Boringcoins hay en la red entera. Empezaremos una variable total en 0, y después un bucle `for` pasará por cada valor de los pares clave-valor del diccionario `accounts`. Después de sumar cada valor a total, imprimiremos la cantidad total de Boringcoins:

```
total = 0
```

```
for amount in accounts.values():
```

```
    total += amount
```

```
print("Total Boringcoins:", total)
```

Al ejecutar el programa, la salida será algo así:

```
{'Al Sweigart': 999058553, 'Miles Bron': 38283, 'not_a_scammer':  
48441,
```

```
'some hacker': 44429, 'Tech Bro': 53424, 'Claire Debella': 54443,
```

```
'Credulous Journalist': 50408, 'Birdie Jay': 36832, 'Carol': 82867,  
'Mark Z.':
```

```
68650, 'Bob': 37920, 'Andi Brand': 57218, 'Eve': 88296, 'Al  
Sweigart sock
```

```
#27': 78080, 'Tax evader': 40937, 'Duke Cody': 17544, 'Lionel  
Toussaint':
```

```
54650, 'some scammer': 2694, 'Alice': 44503, 'David': 41828}
```

```
Total Boringcoins: 1000000000
```

El total es 1000000000, lo cual tiene sentido porque esa es la

cantidad de Boringcoins extraídas antes de empezar.

Paso 2: hacer transacciones

El siguiente programa, `addBoringcoinTransaction.py`, añade filas adicionales a la hoja de cálculo de Google del *blockchain* para añadir nuevas transacciones. Lee tres argumentos de línea de comandos de la lista en `sys.argv`: el emisor, el receptor y la cantidad. Por ejemplo, si ejecutases esto desde el terminal:

```
python addBoringcoinTransaction.py "Al Sweigart" Eve 2000
```

el programa accedería a la hoja de cálculo de Google, añadiría una fila en blanco al final y la rellenaría con los valores ‘Al Sweigart’, ‘Eve’ y ‘2000’. Ten en cuenta que, en el terminal, hay que poner cualquier argumento de línea de comandos que contenga un espacio entre comillas dobles, como “Al Sweigart”, porque, si no, el terminal pensará que se trata de dos argumentos separados.

El inicio de `addBoringcoinTransactions.py` comprueba los argumentos de línea de comandos y asigna variables al emisor, al receptor y a la cantidad basándose en ellos:

```
import sys, ezsheets
```

```
if len(sys.argv) < 4:
```

```
    print('Usage: python addBoringcoinTransaction.py sender recipient  
    amount')
```

```
    sys.exit()
```

```
# Obtiene información sobre la transacción de los argumentos de  
línea de comandos:
```

```
sender, recipient, amount = sys.argv[1:]
```

No hace falta convertir `amount` de cadena a entero porque lo escribiremos como cadena en la hoja de cálculo.

A continuación, `EZSheets` se conecta con la hoja de cálculo de Google que contiene el *blockchain* falso y selecciona la primera hoja

(en el índice 0). Observa que no tienes permiso para editar la hoja de cálculo Boringcoin, así que tendrás que abrir la URL en un navegador web con sesión iniciada en tu cuenta de Google y seleccionar Archivo > Hacer una copia para copiarla en tu cuenta. Después, sustituye la cadena '<https://autbor.com/boringcoin>' por la cadena de la URL de tu hoja de cálculo de Google:

Cambia esta URL por tu copia de la hoja de cálculo de Google, o de lo contrario

recibirás un error indicando que no tienes permiso.

```
ss = ezsheets.Spreadsheet('https://autbor.com/boringcoin')
```

```
sheet = ss.sheets[0]
```

Por último, deberías obtener el número de filas de la hoja, aumentarlas de una en una y rellenar las columnas de dichas filas con datos del emisor, el receptor y la cantidad:

Añade una o varias filas a la hoja para nuevas transacciones:

```
sheet.rowCount += 1
```

```
sheet[1, sheet.rowCount] = sender
```

```
sheet[2, sheet.rowCount] = recipient
```

```
sheet[3, sheet.rowCount] = amount
```

Ahora, cuando ejecutes `python addBoringcoinTransaction.py "Al Sweigart"` Eve 2000 desde el terminal, la hoja de cálculo de Google tendrá una nueva fila con Al Sweigart, Eve y 2000 añadidos al final. Puedes volver a ejecutar el programa `auditBoringcoin.py` para ver los saldos actualizados para todos los de la red de criptomonedas.

El uso de Hojas de cálculo de Google para nuestra estructura de datos *blockchain* es irresponsable, propenso a errores y está abocado a una catástrofe de seguridad. Esto lo equipara con la mayoría de los productos *blockchain* comercializados. ¡No te lo pierdas! ¡Ponte en contacto conmigo para aprovechar esta oferta limitada y comprar Boringcoin antes de que se derrumbe la estafa piramidal!

Trabajar con las cuotas de Hojas de cálculo de Google

Dado que Hojas de cálculo de Google está en línea, podemos compartir fácilmente hojas entre varios usuarios que pueden acceder a ellas al mismo tiempo. Sin embargo, esto también significa que leer y actualizar las hojas será más lento que hacer lo mismo con archivos de Excel almacenados localmente en nuestro disco duro. Además, Hojas de cálculo de Google limita el número de operaciones de lectura y escritura que podemos hacer.

Según las directrices para desarrolladores de Google, los usuarios están limitados a crear 250 hojas de cálculo nuevas al día y las cuentas gratuitas de Google pueden realizar unos pocos cientos de solicitudes por minuto. Encontrarás los límites de uso de Google en <https://developers.google.com/sheets/api/limits>. Intentar exceder esta cuota provocará un excepción `googleapiclient.errors.HttpError`. EZSheets capturará automáticamente esta excepción y reintentará la solicitud. Cuando sucede esto, las llamadas a las funciones para leer o escribir datos pueden llevar segundos (incluso uno o dos minutos) antes de volver. Si la solicitud sigue fallando (lo cual puede pasar si otro *script* con las mismas credenciales también está haciendo solicitudes), EZSheets volverá a lanzar la excepción.

Esto significa que, ocasionalmente, tus llamadas a los métodos de EZSheets pueden tardar varios segundos en volver. Si quieres ver tu uso de la API o aumentar tu cuota, ve a la página IAM y administración > Cuotas y límites del sistema en <https://console.developers.google.com/iam-admin/quotas> para saber más sobre los pagos para aumentar las cuotas. Si prefieres lidiar tú mismo con la excepción `HttpError`, puedes establecer `ezsheets.IGNORE_QUOTA` en `True` para que los métodos de EZSheets lancen estas excepciones cuando las encuentren.

Resumen

Hojas de cálculo de Google es una aplicación de hojas de cálculo popular que se ejecuta en un navegador. Con el paquete de terceros EZSheets, podemos descargar, crear, leer y modificar hojas de cálculo. EZSheets representa las hojas de cálculo como objetos Spreadsheet y cada uno de ellos contienen una lista ordenada de

objetos Sheet. Cada hoja tiene filas y columnas de datos que podemos leer y actualizar de muchas maneras.

Aunque Hojas de cálculo de Google hace que compartir datos y editar cooperativamente sea fácil, su principal desventaja es la velocidad: hay que actualizar las hojas de cálculo con solicitudes web, lo cual puede tardar unos segundos en ejecutarse. Pero, para la mayoría de los fines, esta restricción de velocidad no afectará a los *scripts* de Python que usen EZSheets. Hojas de cálculo de Google también limita la frecuencia con la que se puede hacer cambios.

Puedes consultar la documentación completa de las funciones de EZSheets en <https://ezsheets.readthedocs.io/>.

Preguntas prácticas

1. ¿Qué tres archivos hacen falta para que EZSheets acceda a Hojas de cálculo de Google?
2. ¿Qué dos tipos de objetos tiene EZSheets?
3. ¿Cómo se crea un archivo de Excel desde una hoja de cálculo de Google?
4. ¿Cómo se crea una hoja de cálculo de Google a partir de un archivo de Excel?
5. La variable `ss` contiene un objeto `Spreadsheet`. ¿Qué código leerá datos de la celda B2 de una hoja llamada `Students`?
6. ¿Cómo encontramos las letras para la columna 999?
7. ¿Cómo podemos averiguar cuántas filas y columnas tiene una hoja?
8. ¿Cómo borramos una hoja de cálculo? ¿Es esta eliminación permanente?
9. ¿Qué funciones crearán un nuevo objeto `Spreadsheet` y un nuevo objeto `Sheet`, respectivamente?
10. ¿Qué pasaría si, al hacer solicitudes frecuentes de lectura y

escritura con EZSheets, excedieses la cuota de tu cuenta de Google?

Programas de práctica

Para practicar, escribe programas que hagan las siguientes tareas.

Descargar datos de Formularios de Google

Ya he mencionado antes que Formularios de Google permite crear formularios en línea sencillos que facilitan la recogida de información. Los datos que se introducen en un formulario se guardan en una hoja de Hojas de cálculo de Google. Para este proyecto, escribe un programa que descargue automáticamente la información que han enviado los usuarios. Ve a <https://docs.google.com/forms/> y crea un formulario en blanco. Añade campos para pedir al usuario su nombre y dirección de correo electrónico y haz clic en Enviar, en la parte superior derecha de la pantalla, para obtener un enlace para el formulario. Mete unas cuantas respuestas de ejemplo en el formulario.

En la pestaña Respuestas, haz clic en el botón verde Crear hoja de cálculo para crear una hoja con las respuestas enviadas por los usuarios. Deberías ver tus respuestas de ejemplo en las primeras filas de la hoja. Después, escribe un programa de Python con EZSheets para recoger una lista de las direcciones de correo electrónico de la hoja.

Convertir hojas de cálculo a otros formatos

Puedes usar Hojas de cálculo de Google para convertir un archivo de hoja de cálculo en otros formatos. Escribe un *script* que pase un archivo enviado a `upload()`. Una vez cargada la hoja de cálculo en Hojas de cálculo de Google, descárgala usando `downloadAsExcel()`, `downloadAsODS()` u otra función similar para crear una copia de la hoja en esos formatos.

Buscar errores en una hoja de cálculo

Después de un largo día en la oficina de contabilidad, he terminado

una hoja de cálculo con todos los totales y la he subido a Hojas de cálculo de Google. Está visible públicamente, pero no es editable. Puedes obtener la hoja con este código:

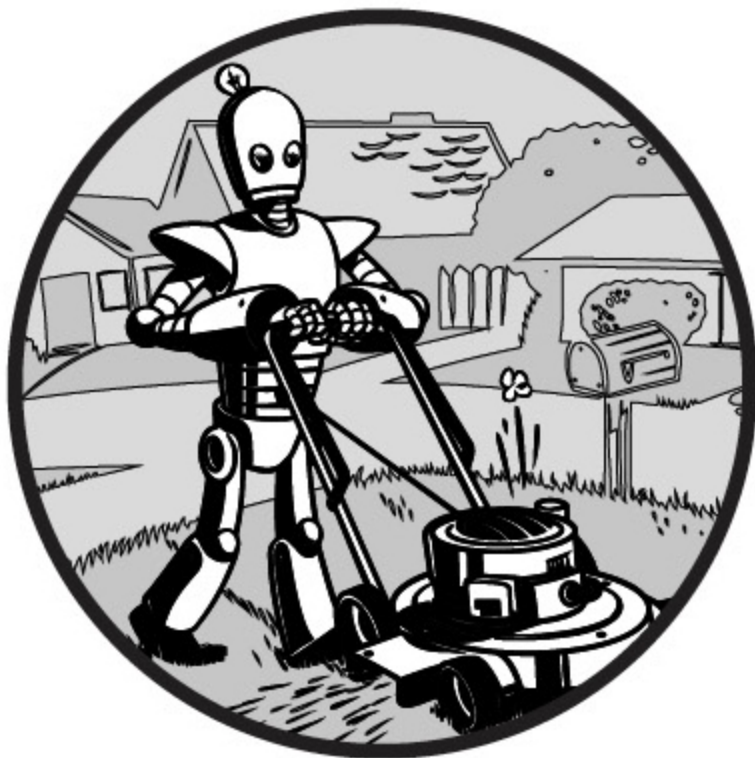
```
> > > import ezsheets
```

```
> > > ss =
```

```
ezsheets.Spreadsheet('1jDZEdvSIh4TmZxccyy0ZXrH-  
ELlrwq8_YYiZrEOB4jg')
```

Abre la hoja en tu navegador yendo a https://docs.google.com/spreadsheets/d/1jDZEdvSIh4TmZxccyy0ZXrH-ELlrwq8_YYiZrEOB4jg. Las columnas de la primera hoja de la hoja de cálculo son BEANS PER JAR, JARS y TOTAL BEANS. La columna TOTAL BEANS es el producto de los números en las columnas BEANS PER JAR y JARS. Sin embargo, hay un error en una de las 15.000 filas de la hoja. Son demasiadas filas que comprobar a mano. Por suerte, puedes escribir un *script* que compruebe los totales.

Una pista: puedes acceder a las celdas individuales de una fila con `ss.sheets[0].getRow(NúmFila)`, donde `ss` es el objeto `Spreadsheet` y `NúmFila` el número de fila. Recuerda que los números de fila de Hojas de cálculo de Google empiezan en 1, no en 0. Los valores de celda serán cadenas, así que tendrás que convertirlos a enteros para que tu programa pueda trabajar con ellos. La expresión `int(ss.sheets[0].getRow(2)[0]) * int(ss.sheets[0].getRow(2)[1]) == int(ss.sheets[0].getRow(2)[2])` se evalúa como `True` si la fila 2 tiene el total correcto. Pon este código en un bucle para identificar qué fila tiene el total incorrecto.



Es probable que tú uses hojas de cálculo de Excel o Google para organizar información, pero la mayoría del software almacena sus datos en aplicaciones llamadas “bases de datos”. Las bases de datos facilitan a los programas la recuperación de los datos específicos que queremos. Si tuvieses una hoja de cálculo o archivo de texto de gatos y quisieras encontrar el color del pelo de una gata llamada Zophie, podrías pulsar **Control-B** y buscar Zophie, pero ¿y si quisieras encontrar el color de todos los gatos que pesen entre 3 y 5 kilos y hayan nacido antes de octubre de 2023? Incluso con las expresiones regulares del capítulo 9, esto sería difícil de codificar.

Las bases de datos permiten realizar búsquedas complejas como esta, escrita en el minilenguaje SQL (*Structured Query Language*, lenguaje de consultas estructurado). Verás que el término SQL se utiliza para referirse tanto al lenguaje para las operaciones con bases de datos como a las bases de datos que entienden este lenguaje. Este capítulo te introduce en el mundo de SQL y otros conceptos de base de datos usando SQLite, una base de datos ligera incluida con Python.

SQLite es el software de base de datos más desplegado, ya que funciona en todos los sistemas operativos y es lo bastante pequeño para incrustarlo dentro de otras aplicaciones. Al mismo tiempo, las simplificaciones de SQLite la hacen claramente distinta de otras bases de datos. Mientras que el software de bases de datos grandes, como PostgreSQL, MySQL, Microsoft SQL Server y Oracle, está pensado para ejecutarse en servidores físicos específicos a los que se accede a través de una red, SQLite almacena toda la base de datos en un único archivo de tu ordenador.

Aunque ya estés familiarizado con las bases de datos SQL, SQLite tiene algunas peculiaridades, así que te conviene leer este capítulo para ver cómo sacarle el máximo partido. Puedes ver la documentación en línea de SQLite en <https://sqlite.org/docs.html> y la del módulo sqlite3 de Python en <https://docs.python.org/3/library/sqlite3.html>.

Hojas de cálculo vs. bases de datos

Vamos a pensar en las similitudes y diferencias entre hojas de cálculo y bases de datos. En una hoja de cálculo, las filas contienen registros individuales, mientras que las columnas representan el tipo de datos guardados en los campos de cada registro. Por ejemplo, la figura 16.1 es una hoja de cálculo con algunos de mis gatos. Las columnas muestran el nombre, el cumpleaños, el color y el peso (en kilos) de cada gato.

	A	B	C	D	E
1	Nombre	Fecha de nacimiento	Pelaje	Peso Kg	
2	Zophie	24/01/2021	negro	5,6	
3	Miguel	24/12/2016	siamés	6,2	
4	Jacob	20/02/2022	naranja y blanco	5,5	
5	Gumdrop	23/08/2020	banco	6,4	
6	Hector	15/03/2022	naranja y blanco	7,5	
7	Night	16/03/2019	esmoquin	6,5	
8	Trey	07/07/2022	tricolor	5,8	
9	Maverick	04/12/2024	siamés	5,6	
10	Powder	30/04/2022	pardo	5	
11	Rita	06/11/2022	pardo	6	
12	Toby	17/05/2021	negro	6,8	
13	Sprinkles	30/07/2013	esmoquin	7	
14	Sadie	06/08/2016	marrón	8	
15	Thea	13/01/2022	marrón	5,9	

Figura 16.1. Una hoja de cálculo almacena registros de datos como filas con una estructura de columnas establecida.

Podemos almacenar esta misma información en una base de datos. Imagina una tabla de una base de datos como una hoja de cálculo. Una base de datos puede contener una o varias tablas, que tienen columnas de diferentes propiedades para cada registro, también llamado fila o entrada. Las bases de datos como SQLite se denominan “bases de datos relacionales”, lo que significa que la base puede contener varias tablas relacionadas entre sí, como veremos más adelante.

Tanto las hojas de cálculo como las bases de datos etiquetan los datos que contienen. Una hoja de cálculo etiqueta automáticamente las columnas con letras y las filas con números. Además, el ejemplo de la hoja de los gatos usa la primera fila para dar a las columnas nombres descriptivos. Cada una de las filas siguientes representa exactamente un gato. En una base de datos SQL, las tablas suelen tener una columna ID para la clave primaria de cada registro: un entero único que puede identificar inequívocamente el registro. En

SQLite, esta columna se llama rowid, y SQLite la añade automáticamente a tus tablas.

Borrar una fila de una hoja de cálculo sube todas las filas que hay debajo, cambiando su número de fila. Pero el ID de clave primaria de un registro de una base de datos es único y no cambia, lo cual es útil en muchas situaciones. ¿Qué pasaría si cambio de nombre a un gato o sube o baja de peso? ¿Y si quisiéramos reordenar las filas por orden alfabético del nombre de los gatos? Cada gato necesita un número de identificación único que se mantenga al margen de cómo cambien los datos. Podríamos añadir una columna ID a nuestra hoja de cálculo para simular la columna rowid de una tabla de SQLite. Este valor ID se mantendrá igual, aunque borremos o movamos filas por la hoja de cálculo, como muestra la figura 16.2, donde los gatos con ID entre 5 y 10 han sido eliminados.

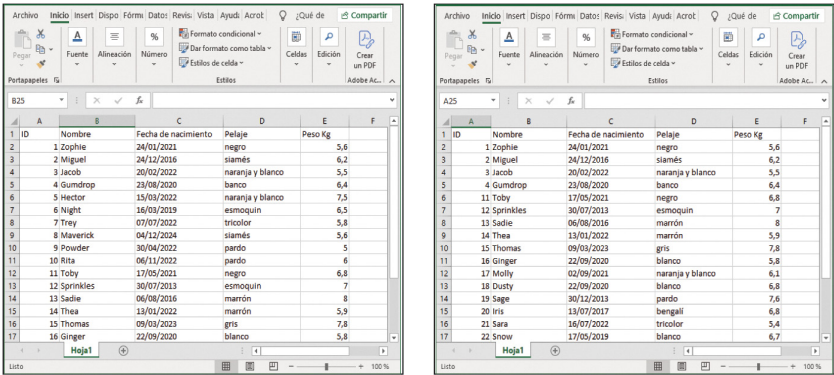


Figura 16.2. El número de la fila ID, a diferencia de los números de fila de la hoja de cálculo, ofrece un identificador único para cada registro (izquierda) que permanece igual, aunque los gatos con ID entre 5 y 10 se borren (derecha).

Hay otra manera en la que la gente usa las hojas de cálculo que no se parece en nada a cómo las bases de datos almacenan la información. Las hojas de cálculo pueden servir como plantillas para formularios en vez de para almacenar datos en filas. Es posible que hayas visto hojas de cálculo como la de la figura 16.3.

Hora	Personal n.º 1	Personal n.º 2	Personal n.º 3	Personal n.º 4
Lunes				
8 am - 12:30 pm	William	Emily	Sophia	Emma
12:30 pm - 5 pm	Benjamin	Sophia	Ava	Michael
5 pm - 9:30 pm	Joshua	Olivia	David	Michael
Martes				
8 am - 12:30 pm	Matthew	Olivia	David	Emma
12:30 pm - 5 pm	Benjamin	Joshua	William	Mia
5 pm - 9:30 pm	Emily	David	Mia	Michael
Miércoles				
8 am - 12:30 pm	Olivia	Sophia	David	Mia
12:30 pm - 5 pm	Benjamin	Joshua	Michael	Emily
5 pm - 9:30 pm	Emily	Emily	Emily	Emily
Jueves				

Figura 16.3. Una hoja de cálculo con mucho formato y un tamaño fijo no suele ser adecuada para una base de datos.

Estas hojas de cálculo tienden a tener mucho formato, con colores de fondo, celdas combinadas y distintas fuentes para quedar bien a ojos humanos. Aunque las hojas de cálculo con datos basados en filas se pueden expandir infinitamente hacia abajo cuando se añaden datos nuevos, estas hojas suelen tener un tamaño fijo y un diseño para rellenar huecos. A menudo, están pensadas para que las miren o impriman humanos y no para que un programa de Python extraiga datos de ellas.

Las bases de datos no son agradables visualmente; solo contienen datos puros y duros. Y lo que es más importante: mientras que las hojas de cálculo te dan flexibilidad para poner cualquier dato en cualquier celda, las bases de datos tienen una estructura más estricta para facilitar al software la recuperación de datos. Si quieres que tus datos queden como en el ejemplo de la figura 16.3, lo mejor sería guardarlos en un archivo JSON o dejarlos en Excel u hojas de cálculo de Google para usar el módulo `openpyxl` del capítulo 14 y la biblioteca `EZSheets` del capítulo 15.

SQLite vs. otras bases de datos SQL

Si estás acostumbrado a trabajar con otras bases de datos SQL, es posible que te estés preguntando en qué se diferencia SQLite. En resumidas cuentas, SQLite ofrece equilibrio entre sencillez y potencia. Es una base de datos relacional completa que usa SQL para leer y escribir cantidades masivas de datos, pero se ejecuta dentro de un programa de Python y opera en un único archivo. El programa importa el módulo `sqlite3` igual que importaría `sys`, `math` o cualquier otro módulo de la librería estándar de Python.

Estas son las principales diferencias entre SQLite y otro software de base de datos:

- Las bases de datos SQLite están guardadas en un solo archivo, que se puede mover, copiar o hacer una copia de seguridad como con cualquier otro archivo.
- SQLite puede ejecutarse en ordenadores con pocos recursos, como sistemas embebidos o portátiles viejos.
- SQLite no tiene servidor; no requiere que se ejecute constantemente en segundo plano una aplicación de servidor en nuestro portátil o hardware. No hay implicadas conexiones de red.
- Desde la perspectiva de los usuarios, SQLite no requiere instalación ni configuración. Forma parte del programa de Python.
- Para un rendimiento más rápido, las bases de datos SQLite pueden existir íntegramente en memoria y guardarse en un archivo antes de que termine el programa.
- Aunque las columnas de SQLite tienen tipos de datos, como números y texto, igual que otras bases de datos SQL, SQLite no fuerza estrictamente el tipo de datos de una columna.
- No hay configuración de permisos ni roles de usuarios en SQLite. SQLite no tiene declaraciones `GRANT` ni `REVOKE` como otras bases de datos SQL.
- SQLite es software de dominio público; puedes usarla comercialmente o como quieras sin restricciones.

La principal desventaja de SQLite es que no puede manejar con

eficiencia cientos o miles de operaciones de escritura simultáneas (por ejemplo, en una *app* web de redes sociales). Aparte de eso, SQLite es igual de potente que cualquier base de datos, capaz de manejar de manera fiable varios GB o incluso TB de datos, además de operaciones de lectura simultáneas, con rapidez y facilidad.

SQLite no se vende tanto como competidor de otro software de base de datos, sino como la competencia de la función `open()` para trabajar con archivos de texto (o los archivos JSON, XML y CSV que veremos en el capítulo 18). Si tu programa necesita la capacidad de almacenar y recuperar rápidamente grandes cantidades de datos, SQLite es mejor opción que JSON o las hojas de cálculo.

Crear bases de datos y tablas

Vamos a empezar a crear nuestra primera base de datos y tabla con SQL. SQL es un minilenguaje con el que podemos trabajar dentro de Python, como `regex` para las expresiones regulares. Al igual que las expresiones regulares, las instrucciones SQL están escritas como valores de cadena de Python. E igual que podíamos escribir nuestro propio código Python para hacer búsquedas de patrones de texto con expresiones regulares, podemos escribir código Python personalizado para buscar datos en diccionarios y listas de Python. Pero escribir expresiones regulares e instrucciones SQL hace estas tareas mucho más fáciles a largo plazo, aunque de primeras requieran aprender nuevas habilidades. Vamos a ver cómo escribir instrucciones que creen tablas en una base de datos nueva.

Crearemos una base de datos SQLite de muestra en un archivo llamado `example.db` para guardar información sobre gatos. Para crear la base de datos, primero importaremos el módulo `sqlite3`. (El 3 es de la versión principal 3 de SQLite, no tiene nada que ver con Python 3). Una base de datos SQLite está en un solo archivo, cuyo nombre puede ser cualquier cosa, pero por convención le pondremos una extensión `.db`. La extensión `.sqlite` también se usa habitualmente.

Una base de datos puede contener múltiples tablas y cada tabla debería guardar un tipo particular de dato. Por ejemplo, una tabla podría contener registros de gatos y otra, registros de las vacunas

que se ha puesto a los gatos de la otra tabla. Puedes pensar en una tabla como en una lista de tuplas, donde cada tupla es una fila de la tabla. La tabla de gatos es, esencialmente, lo mismo que [(‘Zophie’, ‘2021-01-24’, ‘black’, 5.6), (‘Colin’, ‘2016-12-24’, ‘siamese’, 6.2), ..., etc.

Vamos a crear una base de datos y después una tabla para los datos de los gatos, donde insertaremos algunos registros, leeremos los datos y cerraremos la conexión con la base de datos.

Conectarse a bases de datos

El primer paso para escribir código SQLite es obtener un objeto `Connection` para el archivo de la base de datos llamando a `sqlite3.connect()`. Introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > import sqlite3  
  
> > > conn = sqlite3.connect('example.db',  
isolation_level=None)
```

El primer argumento de la función puede ser una cadena con el nombre de archivo o un objeto `pathlib.Path` para el archivo. Si este nombre de archivo no pertenece a una base de datos SQLite existente, la función crea un nuevo archivo con una base de datos en blanco. Por ejemplo, `sqlite3.connect('example.db', isolation_level = None)` conecta con un archivo de base de datos llamado `example.db` en el directorio de trabajo actual. Si no existe, la función lo crea en blanco. Si el archivo existe, pero no es una base de datos SQLite, Python lanza una excepción `sqlite3.DatabaseError: file is not a database` al intentar ejecutar instrucciones. El apartado “Comprobar la validez de una ruta” del capítulo 10 explica cómo usar el método `exists()` de `Path` y la función `os.path.exists()` para decir al programa si un archivo existe o no.

El argumento `isolation_level=None` hace que la base de datos use el modo *autocommit*. Esto nos evita tener que escribir llamadas al método `commit()` después de cada llamada al método `execute()`.

La función `sqlite3.connect()` devuelve un objeto `Connection`, que

guardaremos en una variable llamada `conn` para estos ejemplos. Cada objeto `Connection` se conecta a un archivo de base de datos SQLite. Naturalmente, puedes elegir cualquier nombre de variable que quieras para este objeto `Connection`, pero deberías usar nombres más descriptivos si tu programa abre varias bases de datos al mismo tiempo. Para programas pequeños que se conectan solo a una base, el nombre `conn` es fácil de escribir y lo bastante descriptivo. (Podría ser más corto todavía, pero podría confundirse con “consola”, “contenido” u otros nombres de variables).

Cuando el programa termina con la base de datos, llama a `conn.close()` para cerrar la conexión. El programa también cierra la conexión automáticamente cuando acaba.

Crear tablas

Después de conectar a una base de datos nueva en blanco, crea una tabla con una instrucción `CREATE TABLE SQL`. Para ejecutar instrucciones SQL, hay que llamar al método `execute()` de los objetos `Connection`. Pasa a este método `conn.execute()` una cadena de la instrucción:

```
> > import sqlite3
```

```
> > conn = sqlite3.connect('example.db',  
isolation_level=None)
```

```
> > conn.execute('CREATE TABLE IF NOT EXISTS cats (name  
TEXT NOT NULL,
```

```
birthdate TEXT, fur TEXT, weight_kg REAL) STRICT')
```

```
<sqlite3.Cursor object at 0x00000201B2399540>
```

Por convención, las palabras clave de SQL, como `CREATE` o `TABLE`, se escriben en mayúsculas, aunque no es obligatorio: la instrucción ‘create table if not exists cats (name text not null, birthdate text, fur text, weight_kg real) strict’ funcionará igual. Los nombres de la tabla y las columnas tampoco distinguen mayúsculas y minúsculas, pero la convención es escribirlos en minúscula y separar palabras con guiones bajos, como en `weight_kg`.

La declaración `CREATE TABLE` lanza una excepción `sqlite3.OperationalError: table cats already exists` si intentas crear una tabla que ya existe sin la parte `IF NOT EXISTS`. Incluir esta parte es una forma rápida de evitar tropezarte con esta excepción, y casi siempre te convendrá añadirla a tus instrucciones `CREATE TABLE`.

En nuestro ejemplo, seguimos las palabras clave `CREATE TABLE IF NOT EXISTS` con el nombre `cats` para la tabla. Después del nombre de la tabla hay un par de paréntesis que contienen los nombres de columna y tipos de datos.

Definir tipos de datos

Has seis tipos de datos en SQLite:

- NULL**: Análogo al `None` de Python.
- INT** o **INTEGER**: Análogo al tipo `int` de Python.
- REAL**: Una referencia al término matemático “número real”; análogo al tipo `float` de Python.
- TEXT**: Análogo al tipo `str` de Python.
- BLOB**: Abreviatura de *Binary Large Object*, objeto binario grande; análogo al tipo `bytes` de Python y útil para almacenar archivos enteros en una base de datos.

SQLite tiene sus propios tipos de datos porque no se creó solo para Python; otros lenguajes de programación también pueden interactuar con bases de datos SQLite.

A diferencia de otro software de bases de datos SQL, SQLite no es estricto con el tipo de datos de sus columnas. Esto significa que, por defecto, SQLite guardará alegremente la cadena ‘Hello’ en una columna `INTEGER` sin lanzar una excepción. Pero tampoco es que los tipos de datos de SQLite estén de adorno; SQLite cambia, si es posible, los datos al tipo de datos de la columna, una función llamada “afinidad de tipos”. Por ejemplo, si añadimos la cadena ‘42’ a una columna `INTEGER`, SQLite guardará automáticamente el valor

como el entero 42 porque la columna tiene afinidad de tipos para enteros. Sin embargo, si añadimos la cadena ‘Hello’ a una columna INTEGER, SQLite la guardará tal cual (sin dar error) porque, pese a la afinidad de tipos, ‘Hello’ no se puede convertir en un entero.

La palabra clave STRICT habilita el modo estricto para esta tabla. En dicho modo, cada columna debe recibir un tipo de datos y SQLite lanzará una excepción `sqlite3.IntegrityError` si intentamos introducir datos del tipo incorrecto en la tabla. SQLite seguirá cambiando automáticamente el tipo de datos de la columna cuando se pueda: insertar ‘42’ en una columna INTEGER insertará el entero 42, pero la cadena ‘Hello’, que no se puede cambiar a entero, lanzará una excepción esta vez. Te recomiendo encarecidamente usar el modo estricto porque puede avisarte pronto de fallos provocados por introducir datos incorrectos en tu tabla.

SQLite añadió la palabra clave STRICT en la versión 3.37.0, que es la que usan Python 3.11 y posterior. Las versiones anteriores no conocen este modo y darán un error de sintaxis si intentas utilizarlo. Siempre puedes comprobar la versión de SQLite que está usando Python examinando la variable `sqlite3.sqlite_version`, que será algo así:

```
> > > import sqlite3  
  
> > > sqlite3.sqlite_version  
  
‘3.xx.xx’
```

SQLite no tiene tipo de datos booleano, así que utiliza INTEGER para los datos booleanos; puedes usar 1 para representar True y 0 para representar False. SQLite tampoco tiene tipos de datos de fecha, hora o fecha y hora. En su lugar, puedes usar el tipo de datos TEXT para guardar una cadena en uno de los formatos de la tabla 16.1.

Tabla 16.1. Formatos recomendados para fechas, horas y fechas y horas en SQLite.

Ejemplo
~~2025-MM-DD~~

```
2025-10-11 16:38:00M:SS  
2025-10-11 16:38:00M:SS.SSS  
16:30M:SS  
16:30M:SS.SSS
```

La parte NOT NULL de name TEXT NOT NULL especifica que el valor None de Python no puede almacenarse en la columna name. Es una buena forma de hacer obligatoria una columna de una tabla.

Las tablas de SQLite crean automáticamente una columna rowid que contiene un entero como clave primaria única. Incluso aunque nuestra tabla cats tuviese dos gatos que casualmente se llamen igual, hayan nacido el mismo día, sean del mismo color y pesen lo mismo, rowid nos permitiría distinguirlos.

Listas de tablas y columnas

Todas las bases de datos SQLite tienen una tabla llamada sqlite_schema que recoge metadatos de la base de datos, incluidos todos los de sus tablas. Para ver las tablas de la base de datos SQLite, ejecuta la siguiente consulta:

```
> > > import sqlite3  
  
> > > conn = sqlite3.connect('example.db',  
isolation_level=None)  
  
> > > conn.execute('SELECT name FROM sqlite_schema  
WHERE type="table").fetchall()  
  
[('cats',)]
```

La salida muestra la tabla cats que acabamos de crear. (Explicaré la sintaxis de la declaración SELECT en el apartado “Leer datos de la base de datos”). Para obtener información sobre las columnas de la tabla cats, ejecuta esta consulta:

```
> > > import sqlite3  
  
> > > conn = sqlite3.connect('example.db',  
isolation_level=None)
```



```
> > > conn.execute('PRAGMA TABLE_INFO(cats)').fetchall()
```

```
[(0, 'name', 'TEXT', 1, None, 0), (1, 'birthdate', 'TEXT', 0, None, 0),  
(2,
```

```
'fur', 'TEXT', 0, None, 0), (3, 'weight_kg', 'REAL', 0, None, 0)]
```

Esta consulta devuelve una lista de tuplas, cada una de las cuales describe una columna de la tabla. Por ejemplo, la tupla (1, 'birthdate', 'TEXT', 0, None, 0) nos da la siguiente información sobre la columna birthdate:

- Posición de la columna:** El 1 indica que la columna es la segunda de la tabla. Los números de columna se basan en cero, como los índices de lista de Python, así que la primera columna está en la posición 0.

- Nombre:** 'birthdate' es el nombre de la columna. Recuerda que los nombres de tablas y columnas no distinguen mayúsculas y minúsculas en SQLite.

- Tipo de datos:** 'TEXT' es el tipo de datos SQLite de la columna birthdate.

- Si la columna es NOT NULL:** El 0 significa False, así que la columna no es NOT NULL (es decir, podemos poner valores None en ella).

- Valor predeterminado:** None es el valor predeterminado insertado si no se especifica otro.

- Si la columna es la clave primaria:** El 0 significa False, así que la columna no es una clave primaria.

Fíjate en que la propia tabla `sqlite_schema` no aparece como tabla. Nunca tendrás que modificar esa tabla y, si lo haces, lo más probable es que se corrompa la base de datos y quede ilegible.

Operaciones CRUD de base de datos

CRUD es un acrónimo (en inglés) de las operaciones básicas que

hace la tabla de datos: crear (*create*) datos, leer (*read*) datos, actualizar (*update*) datos y borrar (*delete*) datos. En SQLite, realizamos estas operaciones con declaraciones INSERT, SELECT, UPDATE y DELETE, respectivamente. He aquí unos ejemplos de cada declaración, que después pasaremos como cadenas a `conn.execute()`:

- `INSERT INTO cats VALUES ("Zophie", "2021-01-24", "black", 5.6).`
- `SELECT rowid, * FROM cats ORDER BY fur.`
- `UPDATE cats SET fur = "gray tabby" WHERE rowid = 1.`
- `DELETE FROM cats WHERE rowid = 1.`

La mayoría de las aplicaciones y redes sociales son en realidad interfaces bonitas para bases de datos CRUD. Cuando subimos una foto o respondemos a algo, creamos un registro en una base de datos en alguna parte. Cuando bajamos por la cronología de una red social, estamos leyendo registros de la base de datos. Y, cuando editamos o borramos un *post*, estamos realizando operaciones de edición o eliminación. Siempre que estés aprendiendo sobre una nueva *app*, lenguaje de programación o lenguaje de consulta, usa el acrónimo CRUD para recordar las operaciones básicas que deberías explorar.

Introducir datos en la base de datos

Ahora que tenemos creada la base de datos con una tabla `cats`, vamos a insertar registros para mis mascotas. Tengo unos 300 gatos en mi casa y usar una base de datos SQLite me ayuda a seguirles la pista. Una declaración INSERT puede añadir datos nuevos a una tabla. Escribe el siguiente código en el intérprete de comandos interactivo:

```
> > > import sqlite3
```

```
> > > conn = sqlite3.connect('example.db',  
isolation_level=None)
```

```
> > > conn.execute('CREATE TABLE IF NOT EXISTS cats (name
```

**TEXT NOT NULL, birthdate TEXT,
fur TEXT, weight_kg REAL) STRICT')**

<sqlite3.Cursor object at 0x00000201B2399540 >

**> > > conn.execute('INSERT INTO cats VALUES ("Zophie",
"2021-01-24", "black", 5.6)')**

<sqlite3.Cursor object at 0x00000162842E78C0 >

Esta instrucción INSERT añade una fila nueva a la tabla cats. Entre paréntesis van los valores separados por comas para sus columnas. Los paréntesis son obligatorios en las estructuras INSERT. Observa que, al insertar valores TEXT, he empleado comillas dobles (“) porque ya estoy usando las simples (') para la cadena de la consulta. El módulo sqlite3 utiliza comillas simples o dobles para sus valores TEXT.

Transacciones

Una declaración INSERT empieza con una transacción, que es una unidad de trabajo en una base de datos. Las transacciones deben pasar el test ACID, un concepto de bases de datos que significa que la transacción es:

- Atómica:** La transacción se realiza entera o no se realiza.
- Consistente:** La transacción no infringe restricciones, como las reglas NOT NULL para columnas.
- Aislada:** Una transacción no afecta a otras transacciones.
- Duradera:** Si se completa, los resultados de la transacción se graban en un almacenamiento permanente, como el disco duro.

SQLite es una base de datos que cumple con ACID; incluso ha superado pruebas que simulan la pérdida de energía de un ordenador en medio de una transacción, así que tienes la tranquilidad de que el archivo de datos no se quedará en un estado corrompido e inutilizable. Una instrucción de SQLite insertará los

datos completos en la base de datos o no insertará nada.

Ataques de inyección SQL

Una categoría de técnicas de hackeo llamada “ataques de inyección SQL” puede cambiar tus instrucciones para hacer cosas que no querías. Estas técnicas quedan fuera del alcance de este libro y lo más probable es que no sean un problema para tu código si tus programas no aceptan datos de extraños de Internet. Pero, para evitarlos, usa la sintaxis `?` siempre que hagas referencia a variables al introducir o actualizar datos en tu base de datos. Por ejemplo, para insertar un nuevo registro de gato basado en datos almacenados en variables, yo no insertaría esas variables directamente en la consulta usando Python, así:

```
> > > cat_name = 'Zophie'
```

```
> > > cat_bday = '2021-01-24'
```

```
> > > fur_color = 'black'
```

```
> > > cat_weight = 5.6
```

```
> > > conn.execute(f'INSERT INTO cats VALUES  
("{cat_name}", "{cat_bday}",  
"{fur_color}", "{cat_weight}")')
```

```
<sqlite3.Cursor object at 0x0000022B91BB7C40>
```

Si los valores de estas variables viniesen de entrada de usuario, como en formulario web, un *hacker* podría especificar cadenas que cambiasen el significado de la instrucción. En vez de eso, pondría un `?` en la cadena y pasaría las variables en un argumento de lista después de la cadena de la instrucción:

```
> > > conn.execute('INSERT INTO cats VALUES , , , ,  
[cat_name, cat_bday,  
fur_color, cat_weight]')
```

```
<sqlite3.Cursor object at 0x0000022B91BB7C40>
```

El método `execute()` sustituye los marcadores de posición `?` de la cadena de la instrucción por los valores de las variables después de asegurarse de que no provocarán un ataque de inyección SQL.

Aunque es poco probable que esos ataques se apliquen a tu código, conviene acostumbrarse a usar los marcadores de posición `?` en vez de dar formato a la cadena de instrucción uno mismo.

Leer datos de la base de datos

Cuando haya datos en la base de datos, podremos leerlos con una consulta `SELECT`. Escribe lo siguiente en el intérprete de comandos interactivo para leer datos de la base `example.db`:

```
> > > import sqlite3

> > > conn = sqlite3.connect('example.db',
isolation_level=None)

> > > conn.execute('SELECT * FROM cats').fetchall()

[('Zophie', '2021-01-24', 'black', 5.6)]
```

La llamada al método `execute()` para `SELECT` devuelve un objeto `Cursor`. Para obtener los datos reales, llamamos al método `fetchall()` con este objeto `Cursor`. Cada registro se devuelve como una tupla de la lista de tuplas. Los datos de cada tupla aparecen en el orden de las columnas de la tabla.

En lugar de escribir código Python para pasar por la lista de tuplas nosotros, podemos hacer que SQLite extraiga la información específica que queremos. La instrucción `SELECT` del ejemplo tiene cuatro partes:

- La palabra clave `SELECT`.
- Las columnas que queremos recuperar, donde `*` significa “todas las columnas, menos `rowid`”.
- La palabra clave `FROM`.
- La tabla de la que recuperar datos; en este caso, `cats`.

Si solo quisiéramos las columnas rowid y name de la tabla cats, la consulta sería así:

```
> > > conn.execute('SELECT rowid, name FROM  
cats').fetchall()
```

```
[(1, 'Zophie')]
```

También podemos usar SQL para filtrar los resultados de la consulta, como veremos en el siguiente apartado.

Pasar en bucle por los resultados de la consulta

El método fetchall() devuelve los resultados de la consulta SELECT como una lista de tuplas. Un patrón de código habitual es usar estos datos en un bucle for para realizar alguna operación con cada tupla. Por ejemplo, descarga el archivo sweigartcats.db de la página web del libro e introduce lo siguiente en el intérprete de comandos interactivo para procesar sus datos:

```
> > > import sqlite3
```

```
> > > conn = sqlite3.connect('sweigartcats.db',  
isolation_level=None)
```

```
> > > for row in conn.execute('SELECT * FROM cats'):
```

```
... print('Row data:', row)
```

```
... print(row[0], 'is one of my favorite cats.')
```

```
.
```

```
Row data: ('Zophie', '2021-01-24', 'gray tabby', 5.6)
```

```
Zophie is one of my favorite cats.
```

```
Row data: ('Miguel', '2016-12-24', 'siamese', 6.2)
```

```
Miguel is one of my favorite cats.
```

```
Row data: ('Jacob', '2022-02-20', 'orange and white', 5.5)
```

Jacob is one of my favorite cats.

--fragmento omitido--

El bucle for puede iterar por las tuplas de datos devueltas por `conn.execute()` sin llamar a `fetchall()`, y el código del cuerpo del bucle for puede operar con cada fila de manera individual porque la variable `row` es una tupla de datos de fila de la consulta. El código puede acceder después a las columnas usando el índice entero de la tupla: 0 para el nombre, 1 para la fecha de nacimiento, etc.

Filtrar datos recuperados

Nuestras consultas SELECT han recuperado todas las filas de la tabla, pero a lo mejor solo queremos las que cumplan determinados criterios de filtrado. Con el archivo `sweigartcats.db`, añade una proposición WHERE a la declaración SELECT para proporcionar parámetros de búsqueda, como que el gato sea negro:

```
> > > import sqlite3
```

```
> > > conn = sqlite3.connect('sweigartcats.db',  
isolation_level=None)
```

```
> > > conn.execute('SELECT * FROM cats WHERE fur =  
"black"]').fetchall()
```

```
[1('Zophie', '2021-01-24', 'black', 5.6), ('Toby', '2021-05-17',  
'black',
```

```
6.8), ('Thor', '2013-05-14', 'black', 5.2), ('Sassy', '2017-08-20',  
'black',
```

```
7.5), ('Hope', '2016-05-22', 'black', 7.6)]
```

En este ejemplo, la proposición WHERE WHERE fur = "black" recuperará solo los datos de los registros que contengan "black" en la columna fur.

SQLite define sus propios operadores para usar en la proposición WHERE, pero son similares a los de Python: =, !=, <, >, <=,

`> =`, `AND`, `OR` y `NOT`. Observa que SQLite usa el operador `=` para decir “es igual que”, mientras que Python utiliza `==` para eso. A cada lado del operador, podemos poner un nombre de columna o un valor literal.

La comparación se hará con cada fila de la tabla. Por ejemplo, para `WHERE fur = “black”`, SQLite hace las siguientes comparaciones:

- Como `fur` es `‘black’` y `‘black’ = ‘black’` es verdadero, SQLite incluye la fila 1 en los resultados.

- Para la fila (2, `‘Miguel’`, `‘2016-12-24’`, `‘siamese’`, 6.2), `fur` es `‘siamese’` y `‘siamese’ = ‘black’` es falso, así que no se incluye esa fila en los resultados.

- Para la fila (3, `‘Jacob’`, `‘2022-02-20’`, `‘orange and white’`, 5.5), `fur` es `‘orange and white’` y `‘orange and white’ = ‘black’` es falso, así que no se incluye esa fila en los resultados.

Y así sucesivamente para cada fila de la tabla `cats`.

Vamos a continuar el ejemplo anterior con una proposición `WHERE` más complicada: `WHERE fur = “black” OR birthdate > = “2024-01-01”`. Vamos también a usar la función `pprint.pprint()` para sacar “en bonito” la lista devuelta:

```
> > > import pprint
```

```
> > > matching_cats = conn.execute(“SELECT * FROM cats  
WHERE fur = “black”
```

```
OR birthdate > = “2024-01-01”).fetchall()
```

```
> > > pprint.pprint(matching_cats)
```

```
[('Zophie', '2021-01-24', 'black', 5.6),
```

```
('Toby', '2021-05-17', 'black', 6.8),
```

```
('Taffy', '2024-12-09', 'white', 7.0),
```

```
('Hollie', '2024-08-07', 'calico', 6.0),
```


('Lewis', '2024-03-19', 'orange tabby', 5.1),
('Thor', '2013-05-14', 'black', 5.2),
('Shell', '2024-06-16', 'tortoiseshell', 6.5),
('Jasmine', '2024-09-05', 'orange tabby', 6.3),
('Sassy', '2017-08-20', 'black', 7.5),
('Hope', '2016-05-22', 'black', 7.6)]

Todos los gatos de la lista `matching_cats` resultante son negros o nacieron después del 1 de enero de 2024. Observa que la fecha de nacimiento es solo una cadena. Aunque los operadores de comparación como `>` `=` suelen realizar comparaciones alfabéticas en cadenas, también pueden hacer comparaciones temporales, siempre que el formato de la fecha de nacimiento sea AAAA-MM-DD.

El operador `LIKE` permite hacer coincidir solo el principio o el final de un valor, tratando el signo de porcentaje (%) como un comodín. Por ejemplo, `name LIKE "%y"` dará todos los nombres que terminen por 'y', mientras que `name LIKE "Ja%"` dará todos los nombres que empiecen por 'Ja':

```
> > conn.execute('SELECT rowid, name FROM cats WHERE  
name LIKE "%y").fetchall()
```

```
(5, 'Toby'), (11, 'Molly'), (12, 'Dusty'), (17, 'Mandy'), (18, 'Taffy'),  
(25, 'Rocky'), (27,
```

```
'Bobby'), (30, 'Misty'), (34, 'Mitsy'), (38, 'Colby'), (40, 'Riley'), (46,  
'Ruby'), (65,
```

```
'Daisy'), (67, 'Crosby'), (72, 'Harry'), (77, 'Sassy'), (85, 'Lily'), (93,  
'Spunky')]
```

```
> > > conn.execute('SELECT rowid, name FROM cats WHERE  
name LIKE "Ja%").fetchall()
```

```
[(3, 'Jacob'), (49, 'Java'), (75, 'Jasmine'), (80, 'Jamison')]
```

También podemos usar signos de porcentaje al principio y al final de una cadena para buscar texto en cualquier parte entre medio. Por ejemplo, name LIKE “%ob%” dará todos los nombres que incluyan ‘ob’:

```
> > > conn.execute('SELECT rowid, name FROM cats WHERE  
name LIKE "%ob%").fetchall()
```

```
[(3, 'Jacob'), (5, 'Toby'), (27, 'Bobby')]
```

El operador LIKE busca sin distinguir mayúsculas y minúsculas, así que name LIKE “%ob%” también busca ‘%OB%’, ‘%Ob%’ y ‘%oB%’. Para hacer una búsqueda diferenciando mayúsculas y minúsculas, utiliza el operador GLOB y * como comodín:

```
> > > conn.execute('SELECT rowid, name FROM cats WHERE  
name GLOB "*m*").fetchall()
```

```
[(4, 'Gumdrop'), (9, 'Thomas'), (44, 'Sam'), (63, 'Cinnamon'), (75,  
'Jasmine'),
```

```
(79, 'Samantha'), (80, 'Jamison')]
```

Mientras que name LIKE “%m%” busca una m mayúscula o minúscula, name GLOB “*m*” solo la busca en minúscula.

El amplio conjunto de operadores y funcionalidades de SQLite no tiene nada que envidiar a cualquier lenguaje de programación. Puede leer más sobre esto en la documentación de SQLite en https://www.sqlite.org/lang_expr.html.

Ordenar los resultados

Aunque podemos ordenar la lista devuelta por fetchall() llamando al método sort() de Python, es más fácil hacer que SQLite ordene los datos por nosotros añadiendo una proposición ORDER BY a la consulta SELECT. Por ejemplo, si queremos ordenar los gatos por color, podríamos introducir lo siguiente:

```
> > > import sqlite3, pprint
```

```
> > > conn = sqlite3.connect('sweigartcats.db',  
isolation_level=None)
```

```
> > > pprint.pprint(conn.execute('SELECT * FROM cats  
ORDER BY fur').fetchall())
```

```
[('Iris', '2017-07-13', 'bengal', 6.8),  
(('Ruby', '2023-12-22', 'bengal', 5.0),  
(('Elton', '2020-05-28', 'bengal', 5.4),  
(('Toby', '2021-05-17', 'black', 6.8),  
(('Thor', '2013-05-14', 'black', 5.2),  
--fragmento omitido--  
(('Celine', '2015-04-18', 'white', 7.3),  
(('Daisy', '2019-03-19', 'white', 6.0)]
```

Si hay una proposición WHERE en la consulta, la proposición ORDER BY debería ir después. También podemos ordenar las filas basándonos en múltiples columnas. Por ejemplo, si quieres ordenar primero las filas por color y, dentro de cada color, por fecha de nacimiento, ejecuta lo siguiente:

```
> > > cur = conn.execute('SELECT * FROM cats ORDER BY  
fur, birthdate')
```

```
> > > pprint.pprint(cur.fetchall())  
[('Iris', '2017-07-13', 'bengal', 6.8),  
(('Elton', '2020-05-28', 'bengal', 5.4),  
(('Ruby', '2023-12-22', 'bengal', 5.0),  
(('Thor', '2013-05-14', 'black', 5.2),  
(('Hope', '2016-05-22', 'black', 7.6),
```

--fragmento omitido--

('Ginger', '2020-09-22', 'white', 5.8),

('Taffy', '2024-12-09', 'white', 7.0)]

La proposición ORDER BY hace primero una lista con la columna fur, seguida de la columna birthdate, separada por una coma. Por defecto, el orden es ascendente: primero van los valores más pequeños y, después, los más grandes. Para ordenar en orden descendente, añade la palabra clave DESC después de la columna name. También puedes usar la palabra clave ASC para especificar el orden ascendente si quieres que tu consulta sea explícita y legible. Para practicar a usar estas palabras clave, introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > cur = conn.execute('SELECT * FROM cats ORDER BY fur  
ASC, birthdate DESC')
```

```
> > > pprint.pprint(cur.fetchall())
```

('Ruby', '2023-12-22', 'bengal', 5.0),

('Elton', '2020-05-28', 'bengal', 5.4),

('Iris', '2017-07-13', 'bengal', 6.8),

('Toby', '2021-05-17', 'black', 6.8),

('Sassy', '2017-08-20', 'black', 7.5),

--fragmento omitido--

('Mitsy', '2015-05-29', 'white', 5.0),

('Celine', '2015-04-18', 'white', 7.3)]

La salida enumera los gatos por color en orden ascendente (con 'bengal' antes que 'white'). Dentro de cada color, los gatos están ordenados por fecha de nacimiento en orden descendente (con '2023-12-22' antes que '2020-05-28').

Limitar el número de resultados

Si estás interesado en ver solo las primeras filas devueltas por la consulta SELECT, prueba a usar rebanadas de lista de Python para limitar los resultados. Por ejemplo, usa la rebanada [:3 para mostrar solo las tres primeras filas de la tabla cats:

```
> > > import sqlite3
```

```
> > > conn = sqlite3.connect('sweigartcats.db',  
isolation_level=None)
```

```
> > > conn.execute('SELECT * FROM cats').fetchall()[:3] #  
Esto no es eficiente.
```

```
[('Zophie', '2021-01-24', 'gray tabby', 5.6), ('Miguel', '2016-12-24',  
'siamese', 6.2), ('Jacob', '2022-02-20', 'orange and white', 5.5)]
```

Este código funciona, pero no es eficiente. Primero busca todas las filas de la tabla, luego descarta todas menos las tres primeras. Sería más rápido que buscarse solo las tres primeras filas de la base de datos, algo que podemos hacer con una proposición LIMIT:

```
> > > conn.execute('SELECT * FROM cats LIMIT 3').fetchall()
```

```
[('Zophie', '2021-01-24', 'gray tabby', 5.6), ('Miguel', '2016-12-24',  
'siamese', 6.2), ('Jacob', '2022-02-20', 'orange and white', 5.5)]
```

Este código se ejecuta más rápido que el que busca todas las filas, sobre todo en tablas con muchas filas. La proposición LIMIT debe ir después de las proposiciones WHERE y ORDER BY si la consulta SELECT las incluye, como en el siguiente ejemplo:

```
> > > conn.execute('SELECT * FROM cats WHERE  
fur="orange" ORDER BY birthdate LIMIT 4').fetchall()
```

```
[('Mittens', '2013-07-03', 'orange', 7.4), ('Piers', '2014-07-08',  
'orange', 5.2),
```

```
('Misty', '2016-07-08', 'orange', 5.2), ('Blaze', '2023-01-16', 'orange',
```

7.4)]

Hay otras proposiciones que podemos añadir a nuestras consultas SELECT, pero están fuera del ámbito de este capítulo; puedes aprender más sobre ellas con la documentación de SQLite.

Crear índices para una lectura de datos más rápida

En un apartado anterior, hemos ejecutado una consulta SELECT para buscar registros basados en nombres coincidentes. Podemos acelerar esta búsqueda creando un índice con la columna name. Un índice SQL es una estructura de datos que organiza los datos de una columna. Como resultado, las consultas con proposiciones WHERE que usan estas columnas tendrán mejor rendimiento. La desventaja es que el índice ocupa un poco más de espacio de almacenamiento, así que las consultas que inserten o actualicen datos serán ligeramente más lentas porque SQLite tendrá que actualizar también el índice. Si tu base de datos es grande y lees datos de ella con más frecuencia que con la que introduces o actualizas datos, puede merecerte la pena crear un índice. Sin embargo, deberías hacer pruebas para verificar que el índice mejora realmente el rendimiento. Para crear índices en las columnas names y birthdate de la tabla cats, por ejemplo, ejecuta las siguientes instrucciones CREATE INDEX:

```
> > > import sqlite3
```

```
> > > conn = sqlite3.connect('sweigartcats.db',  
isolation_level=None)
```

```
> > > conn.execute('CREATE INDEX idx_name ON cats  
(name)')
```

```
<sqlite3.Cursor object at 0x0000013EC121A040>
```

```
> > > conn.execute('CREATE INDEX idx_birthdate ON cats  
(birthdate)')
```

```
<sqlite3.Cursor object at 0x0000013EC121A040>
```

Los índices requieren nombres y, por convención, les ponemos el de

la columna a la que se aplican, junto con el prefijo `idx_`. Los nombres de índice son globales para toda la base de datos, así que, si la base contiene varias tablas con columnas llamadas `birthdate`, puede que nos interese incluir también la tabla en el nombre del índice, como `idx_cats_birthdate`. Para ver todos los índices que existen para una tabla, comprueba la tabla integrada `sqlite_schema` con una consulta `SELECT`:

```
> > > conn.execute('SELECT name FROM sqlite_schema
WHERE type = "index" AND
tbl_name = "cats").fetchall()

[('idx_name',), ('idx_birthdate',)]
```

Si cambias de opinión o descubres que los índices no mejoran el rendimiento, puedes borrarlos con una instrucción `DROP INDEX`:

```
> > > conn.execute('SELECT name FROM sqlite_schema
WHERE type = "index" AND
tbl_name = "cats").fetchall()

[('idx_birthdate',), ('idx_name',)]
```

```
> > > conn.execute('DROP INDEX idx_name')

<sqlite3.Cursor object at 0x0000013EC121A040>
```

```
> > > conn.execute('SELECT name FROM sqlite_schema
WHERE type = "index" AND
tbl_name = "cats").fetchall()

[('idx_birthdate',)]
```

Para bases de datos pequeñas, con solo unos pocos miles de registros, puedes ignorar los índices tranquilamente, ya que los beneficios que van a aportar son ínfimos. Sin embargo, si te das cuenta de que tus consultas tardan un tiempo considerable, crear índices puede mejorar su rendimiento.

Una vez que hemos insertado filas en una tabla, podemos cambiar una fila con una declaración UPDATE. Por ejemplo, vamos a actualizar el registro (1, 'Zophie', '2021-01-24', 'black', 5.6) para cambiar el color de 'black' a 'gray tabby' en el archivo `sweigartcats.db`:

```
> > > import sqlite3
```

```
> > > conn = sqlite3.connect('sweigartcats.db',  
isolation_level=None)
```

```
> > > conn.execute('SELECT * FROM cats WHERE rowid =  
1').fetchall()
```

```
[('Zophie', '2021-01-24', 'black', 5.6)]
```

```
> > > conn.execute('UPDATE cats SET fur = "gray tabby"  
WHERE rowid = 1')
```

```
<sqlite3.Cursor object at 0x0000013EC121A040>
```

```
> > > conn.execute('SELECT * FROM cats WHERE rowid =  
1').fetchall()
```

```
[('Zophie', '2021-01-24', 'gray tabby', 5.6)]
```

La declaración UPDATE consta de las siguientes partes:

- La palabra clave UPDATE.
- El nombre de la tabla que contiene las filas que se van a actualizar.
- La proposición SET, que especifica la columna que actualizar, así como el valor que hay que poner.
- La proposición WHERE, que especifica qué filas actualizar.

Podemos actualizar varias columnas a la vez separándolas con comas. Por ejemplo, la instrucción `'UPDATE cats SET fur = "black", weight_kg = 6 WHERE rowid = 1'` actualiza el valor de las

columnas fur y weight a “black” y 6, respectivamente.

La instrucción UPDATE actualiza todas las filas en las que la proposición WHERE sea verdadera. Si ejecutásemos la instrucción ‘UPDATE cats SET fur = “gray tabby” WHERE name = “Zophie”’, las actualizaciones se aplicarían a todos los gatos llamados Zophie, lo cual pueden ser más gatos de lo que esperábamos. Por eso, en la mayoría de las instrucciones de actualización, la proposición WHERE utiliza la clave primaria de la columna rowid para especificar un registro individual que actualizar. La clave primaria identifica de manera inequívoca una fila, así que usarla en la proposición WHERE garantiza que solo actualizamos la fila que queríamos.

Es un fallo habitual olvidar la proposición WHERE al actualizar datos. Por ejemplo, si queremos hacer una búsqueda y sustitución para cambiar todos los gatos de color ‘white and orange’ a ‘orange and white’, deberíamos ejecutar lo siguiente:

```
> > > conn.execute(‘UPDATE cats SET fur = “orange and  
white” WHERE fur = “white and orange”’)
```

Si olvidásemos incluir la proposición WHERE, las actualizaciones se aplicarían a todas las filas de la tabla y, de repente, ¡todos los gatos serían de color naranja y blanco!

Para evitar este fallo, incluye siempre una proposición WHERE en tus instrucciones UPDATE, incluso aunque quieras aplicar un cambio a todas las filas. En ese caso, podrías usar WHERE 1. Dado que 1 es el valor que utiliza SQLite para el booleano True, esto dice a SQLite que aplique el cambio a todas las filas. Puede parecer una tontería tener un WHERE 1 al final de la instrucción, pero te permite evitar fallos peligrosos que podrían borrar fácilmente tus datos.

Eliminar datos de la base de datos

Podemos borrar filas de una tabla con una instrucción DELETE. Por ejemplo, para eliminar a Zophie de la tabla cats, ejecuta esto en el archivo sweigartcats.db:

```
> > > import sqlite3
```

```
> > > conn = sqlite3.connect('sweigartcats.db',  
isolation_level=None)
```

```
> > > conn.execute('SELECT rowid, * FROM cats WHERE  
rowid = 1').fetchall()
```

```
[(1, 'Zophie', '2021-01-24', 'gray tabby', 5.6)]
```

```
> > > conn.execute('DELETE FROM cats WHERE rowid = 1')
```

```
<sqlite3.Cursor object at 0x0000020322D183C0>
```

```
> > > conn.execute('SELECT * FROM cats WHERE rowid =  
1').fetchall()
```

```
[]
```

La declaración DELETE tiene las siguientes partes:

- Las palabras clave DELETE FROM.
- El nombre de la tabla que contiene las filas que eliminar.
- La proposición WHERE, que especifica qué filas eliminar.

Como con la declaración INSERT, es crucial incluir siempre una proposición WHERE en las declaraciones DELETE; de lo contrario, borraremos todas las filas de la tabla. Si eso es lo que queremos, podemos usar WHERE 1 para identificar cualquier declaración DELETE sin proposición WHERE como fallo.

Deshacer transacciones

Es posible que a veces quieras ejecutar varias consultas a la vez o que no quieras ejecutar esas consultas, pero no sabrás lo que quieres hacer hasta que hayas ejecutado algunas. Una forma de manejar esta situación es empezar una transacción nueva, ejecutar las instrucciones y, después, confirmar todas las instrucciones ejecutadas para completar la transacción o deshacerlas para que la base de datos quede como si nunca se hubiesen ejecutado. Por lo

general, una transacción nueva empieza y acaba cada vez que llamamos a `conn.execute()` estando conectados a la base de datos SQLite en modo *autocommit*. Sin embargo, también podemos ejecutar una instrucción `BEGIN` para iniciar una transacción de manera que luego podamos completarla llamando a `conn.commit()` o deshacer todas las instrucciones llamando a `conn.rollback()`.

Por ejemplo, vamos a añadir dos gatos nuevos a la tabla `cats` y a deshacer después la transacción para que la tabla no cambie:

```
> > > import sqlite3
```

```
> > > conn = sqlite3.connect('sweigartcats.db',  
isolation_level=None)
```

```
> > > conn.execute('BEGIN')
```

```
<sqlite3.Cursor object at 0x00000219C8BF7C40>
```

```
> > > conn.execute('INSERT INTO cats VALUES ("Socks",  
"2022-04-04", "white", 4.2)')
```

```
<sqlite3.Cursor object at 0x00000219C8BF7C40>
```

```
> > > conn.execute('INSERT INTO cats VALUES ("Fluffy",  
"2022-10-30", "gray", 4.5)')
```

```
<sqlite3.Cursor object at 0x00000219C8BF7C40>
```

```
> > > conn.rollback() # Esto deshace las declaraciones INSERT.
```

```
> > > conn.execute('SELECT * FROM cats WHERE name =  
"Socks").fetchall()
```

```
[]
```

```
> > > conn.execute('SELECT * FROM cats WHERE name =  
"Fluffy").fetchall()
```

```
[]
```

Los nuevos gatos, Socks y Fluffy, no se han insertado en la base de

datos. Por otro lado, si queremos aplicar todas las instrucciones ejecutadas, llamaremos a `conn.commit()` para confirmar los cambios en la base de datos:

```
> > > conn.execute('BEGIN')
```

```
<sqlite3.Cursor object at 0x00000219C8BF7C40>
```

```
> > > conn.execute('INSERT INTO cats VALUES ("Socks",  
"2022-04-04", "white", 4.2)')
```

```
<sqlite3.Cursor object at 0x00000219C8BF7C40>
```

```
> > > conn.execute('INSERT INTO cats VALUES ("Fluffy",  
"2022-10-30", "gray", 4.5)')
```

```
<sqlite3.Cursor object at 0x00000219C8BF7C40>
```

```
> > > conn.commit()
```

```
> > > conn.execute('SELECT * FROM cats WHERE name =  
"Socks").fetchall()
```

```
[('Socks', '2022-04-04', 'white', 4.2)]
```

```
> > > conn.execute('SELECT * FROM cats WHERE name =  
"Fluffy").fetchall()
```

```
[('Fluffy', '2022-10-30', 'gray', 4.5)]
```

Ahora los gatos Socks y Fluffy están registrados en la base de datos.

Hacer copias de seguridad de bases de datos

Una amiga mía estaba una vez haciendo cambios en una base de datos que utilizaba un sitio de comercio en línea especializado en cromos deportivos. Tenía que corregir algunos errores en los nombres de algunos cromos y solo había escrito `UPDATE cards SET name = 'Chris Clemons'` cuando su gato pisó el teclado y pulsó **Intro**. Sin una proposición `WHERE`, la instrucción actualizó los miles de cromos que vendía el sitio web.

Por suerte, tenía copias de seguridad de la base de datos, así que pudo restaurar el estado anterior. (Esto fue muy útil porque volvió a pasar lo mismo exactamente, llevándola a pensar que el gato lo estaba haciendo a propósito).

Si un programa no está en este momento accediendo a la base de datos SQLite, podemos devolverlo a su estado anterior simplemente copiando el archivo de la base. Un programa de Python podría hacerlo llamando a `shutil.copy('sweigartcats.db', 'backup.db')`, como vimos en el capítulo 11. Sin embargo, si el software está constantemente leyendo o actualizando el contenido de la base de datos, tendremos que usar el método `backup()` del objeto `Connection`. Por ejemplo, introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > import sqlite3

> > > conn = sqlite3.connect('sweigartcats.db',
isolation_level=None)

> > > backup_conn = sqlite3.connect('backup.db',
isolation_level=None)

> > > conn.backup(backup_conn)
```

El método `backup()` copia con seguridad el contenido de la base de datos `sweigartcats.db` en el archivo `backup.db` entre medio de las demás instrucciones que se están ejecutando en ella. Ahora que los datos cuentan con una copia de seguridad, el gato puede pisar el teclado las veces que quiera.

Modificar y eliminar tablas

Después de crear una tabla en una base de datos e insertarle filas, puede que queramos renombrar la tabla o sus columnas. También es posible que necesitemos añadir o eliminar columnas o incluso eliminar toda la tabla. Podemos utilizar una instrucción `ALTER TABLE` para todas estas acciones.

Los siguientes ejemplos del intérprete de comandos interactivo empiezan con una nueva copia del archivo `sweigartcats.db`. Vamos a

ejecutar una instrucción ALTER TABLE RENAME para cambiar el nombre de la tabla, cats, por felines:

```
> > > import sqlite3
```

```
> > > conn = sqlite3.connect('sweigartcats.db',  
isolation_level=None)
```

```
> > > conn.execute('SELECT name FROM sqlite_schema  
WHERE type="table").fetchall()
```

```
[('cats',)]
```

```
> > > conn.execute('ALTER TABLE cats RENAME TO felines') #  
Cambia el nombre de la tabla.
```

```
<sqlite3.Cursor object at 0x000001EDDB477C40>
```

```
> > > conn.execute('SELECT name FROM sqlite_schema  
WHERE type="table").fetchall()
```

```
[('felines',)]
```

Para cambiar el nombre de una columna de la tabla, ejecutaremos una instrucción ALTER TABLE RENAME COLUMN. Por ejemplo, vamos a cambiar la columna fur por description:

```
> > > conn.execute('PRAGMA TABLE_INFO(felines)').fetchall()  
[2] # Hace una lista de la tercera columna.
```

```
(2, 'fur', 'TEXT', 0, None, 0)
```

```
> > > conn.execute('ALTER TABLE felines RENAME COLUMN  
fur TO description')
```

```
<sqlite3.Cursor object at 0x000001EDDB477C40>
```

```
> > > conn.execute('PRAGMA TABLE_INFO(felines)').fetchall()  
[2] # Hace una lista de la tercera columna.
```

```
(2, 'description', 'TEXT', 0, None, 0)
```

Para añadir una columna a la tabla, usamos ALTER TABLE ADD

COLUMN. Por ejemplo, vamos a añadir una columna `is_loved` a la tabla `felines` para que contenga un valor booleano. SQLite utiliza 0 para falso y 1 para verdadero; estableceremos el valor predeterminado para `is_loved` en 1:

```
> > > conn.execute('ALTER TABLE felines ADD COLUMN  
is_loved INTEGER DEFAULT 1')
```

```
<sqlite3.Cursor object at 0x000001EDDB477C40>
```

```
> > > import pprint
```

```
> > > pprint.pprint(conn.execute('SELECT * FROM felines  
LIMIT 3').fetchall())
```

```
[('Zophie', '2021-01-24', 'gray tabby', 5.6, 1),
```

```
('Miguel', '2016-12-24', 'siamese', 6.2, 1),
```

```
('Jacob', '2022-02-20', 'orange and white', 5.5, 1)]
```

Resulta que, como quiero a todos mis gatos, no es necesaria la columna `is_loved` porque todas las filas van a contener 1, así que voy a quitarla con una instrucción `ALTER TABLE DROP COLUMN`:

```
> > > conn.execute('PRAGMA TABLE_INFO(felines)').fetchall()  
# Hace una lista de todas las columnas.
```

```
[(0, 'name', 'TEXT', 1, None, 0), (1, 'birthdate', 'TEXT', 0, None, 0),  
(2, 'description', 'TEXT',
```

```
0, None, 0), (3, 'weight_kg', 'REAL', 0, None, 0), (4, 'is_loved',  
'INTEGER', 0, '1', 0)]
```

```
> > > conn.execute('ALTER TABLE felines DROP COLUMN  
is_loved') # Borra la columna.
```

```
<sqlite3.Cursor object at 0x000001EDDB477C40>
```

```
> > > conn.execute('PRAGMA TABLE_INFO(felines)').fetchall()  
# Hace una lista de todas las columnas.
```

```
[(0, 'name', 'TEXT', 1, None, 0), (1, 'birthdate', 'TEXT', 0, None, 0),  
(2, 'description', 'TEXT',  
0, None, 0), (3, 'weight_kg', 'REAL', 0, None, 0)]
```

También se borrarán los datos de la columna eliminada.

Si quieres eliminar toda la tabla, ejecuta una instrucción DROP TABLE:

```
> > > conn.execute("SELECT name FROM sqlite_schema  
WHERE type = "table").fetchall()
```

```
[('felines',)]
```

```
> > > conn.execute("DROP TABLE felines") # Borra la tabla  
entera.
```

```
<sqlite3.Cursor object at 0x000001EDDB477C40>
```

```
> > > conn.execute("SELECT name FROM sqlite_schema  
WHERE type = "table").fetchall()
```

```
[]
```

Intenta limitar la frecuencia con la que cambias tus tablas y columnas, porque también tendrás que actualizar las instrucciones y consultas de tu programa en consecuencia.

Unir varias tablas con claves foráneas

La estructura de las tablas SQLite es bastante estricta; por ejemplo, cada fila tiene un número determinado de columnas. Pero los datos del mundo real son muchas veces más complejos de lo que puede reflejar una sola tabla. En las bases de datos relacionales, podemos almacenar datos complejos en varias tablas y crear entre ellas enlaces llamados “claves foráneas”.

Supongamos que queremos guardar información sobre las vacunas que ponemos a nuestros gatos. No basta con añadir columnas a nuestra tabla cats porque cada gato podría tener una o varias vacunas. Además, para cada vacuna, queremos tener la fecha y el

nombre del veterinario que se la puso. A las tablas SQL no se les da bien almacenar una lista de columnas. No queremos tener un montón de columnas llamadas vaccination1, vaccination2, vaccination3, etcétera, por la misma razón que no conviene tener variables llamadas vaccination1 y vaccination2. Si creamos demasiadas columnas o variables, el código se convierte en un jaleo verboso e ilegible. Si creamos pocas, habrá que actualizar constantemente el programa para ir añadiendo más cuando haga falta.

Siempre que haya una cantidad variable de datos que añadir a una columna, tiene más sentido recoger los datos añadidos como filas en una tabla aparte y luego hacer que esas filas hagan referencia a las de la tabla principal. En nuestra base de datos sweigartcats.db, añade una segunda tabla vaccinations poniendo esto en el intérprete de comandos interactivo:

```
> > > import sqlite3

> > > conn = sqlite3.connect('sweigartcats.db',
isolation_level=None)

> > > conn.execute('PRAGMA foreign_keys = ON')

<sqlite3.Cursor object at 0x000001E730AD03C0>

> > > conn.execute('CREATE TABLE IF NOT EXISTS
vaccinations (vaccine TEXT,
date_administered TEXT, administered_by TEXT, cat_id
INTEGER,
FOREIGN KEY(cat_id) REFERENCES cats(rowid)) STRICT')

<sqlite3.Cursor object at 0x000001CA42767D40>
```

La nueva tabla vaccinations tiene una columna llamada cat_id con un tipo de datos INTEGER. Los valores enteros de esta columna coinciden con los valores de rowid en la tabla cats. Llamamos a la columna cat_id clave foránea porque hace referencia a la clave primaria de otra tabla.

En la tabla cats, Zophie tiene el rowid de 1. Para registrar sus vacunas, insertamos nuevas filas en la tabla vaccinations con un valor cat_id value de 1:

```
> > > conn.execute('INSERT INTO vaccinations VALUES  
("rabies", "2023-06-06", "Dr. Echo", 1)')
```

```
<sqlite3.Cursor object at 0x000001CA42767D40>
```

```
> > > conn.execute('INSERT INTO vaccinations VALUES  
("FeLV", "2023-06-06", "Dr. Echo", 1)')
```

```
<sqlite3.Cursor object at 0x000001CA42767D40>
```

```
> > > conn.execute('SELECT * FROM vaccinations').fetchall()  
[('rabies', '2023-06-06', 'Dr. Echo', 1), ('FeLV', '2023-06-06', 'Dr.  
Echo', 1)]
```

Podríamos registrar vacunas para otros gatos usando su rowid. Si quisiéramos añadir vacunas para Mango, podríamos buscar su rowid en la tabla cats y añadir registros en vaccinations usando ese valor para la columna cat_id:

```
> > > conn.execute('SELECT rowid, * FROM cats WHERE name  
= "Mango").fetchall()
```

```
[(23, 'Mango', '2017-02-12', 'tuxedo', 6.8)]
```

```
> > > conn.execute('INSERT INTO vaccinations VALUES  
("rabies", "2023-07-11", "Dr. Echo", 23)')
```

```
<sqlite3.Cursor object at 0x000001CA42767D40>
```

También podemos realizar un tipo de consulta SELECT llamado unión interna que devuelve las filas vinculadas de ambas tablas. Por ejemplo, escribe lo siguiente en el intérprete de comandos interactivo para recuperar las filas de vaccinations unidas a los datos de cats:

```
> > > conn.execute('SELECT * FROM cats INNER JOIN  
vaccinations ON cats.rowid =
```

vaccinations.cat_id').fetchall()

```
[('Zophie', '2021-01-24', 'gray tabby', 5.6, 'rabies', '2023-06-06',  
'Dr. Echo', 1),
```

```
('Zophie', '2021-01-24', 'gray tabby', 5.6, 'FeLV', '2023-06-06', 'Dr.  
Echo', 1),
```

```
('Mango', '2017-02-12', 'tuxedo', 6.8, 'rabies', '2023-07-11', 'Dr.  
Echo', 23)]
```

Observa que, aunque podríamos hacer de `cat_id` una columna `INTEGER` y usarla como clave foránea sin configurar realmente la sintaxis `FOREIGN KEY(cat_id) REFERENCES cats(rowid)`, las claves foráneas tienen funciones de seguridad que garantizan la consistencia de los datos. Por ejemplo, no podemos insertar ni actualizar un registro de vacunación con un `cat_id` de un gato que no existe. SQLite también te obliga a borrar todos los registros de vacunación de un gato antes de eliminar el gato para no dejar registros “huérfanos” por ahí. Estas funciones de seguridad están deshabilitadas por defecto. Puedes habilitarlas ejecutando la instrucción `PRAGMA` después de llamar a `sqlite3.connect()`:

```
> > > conn.execute('PRAGMA foreign_keys = ON')
```

Las claves foráneas y las uniones tienen funciones adicionales, pero quedan fuera del alcance de este libro.

Bases de datos y copias de seguridad en memoria

Si nuestro programa hace muchas consultas, podemos mejorar significativamente la velocidad de la base de datos utilizando una base de datos en memoria. Estas bases están guardadas íntegramente en la memoria del ordenador, en lugar de en un archivo del disco duro. Esto agiliza muchísimo los cambios. Sin embargo, tenemos que acordarnos de guardar la base de datos en memoria en un archivo con el método `backup()`. Si el programa falla en mitad de la ejecución, perderemos toda la base de datos en memoria igual que perderíamos los valores almacenados en las variables.

El siguiente ejemplo crea una base de datos en memoria y la guarda en una base de datos en el archivo test.db:

```
> > > import sqlite3

> > > memory_db_conn = sqlite3.connect(':memory:',
isolation_level=None) # Crea una base de datos en memoria.

> > > memory_db_conn.execute('CREATE TABLE test (name
TEXT, number REAL)')

<sqlite3.Cursor object at 0x000001E730AD0340>

> > > memory_db_conn.execute('INSERT INTO test VALUES
("foo", 3.14)')

<sqlite3.Cursor object at 0x000001D9B0A07EC0>

> > > file_db_conn = sqlite3.connect('test.db',
isolation_level=None)

> > > memory_db_conn.backup(file_db_conn) # Guarda la base
de datos en test.db.
```

También podemos cargar un archivo de base de datos SQLite en memoria con el método backup():

```
> > > import sqlite3

> > > file_db_conn = sqlite3.connect('sweigartcats.db',
isolation_level=None)

> > > memory_db_conn = sqlite3.connect(':memory:',
isolation_level=None)

> > > file_db_conn.backup(memory_db_conn)

> > > memory_db_conn.execute('SELECT * FROM cats LIMIT
3').fetchall()

[('Zophie', '2021-01-24', 'gray tabby', 5.6), ('Miguel', '2016-12-24',
```

‘siamese’, 6.2), (‘Jacob’, ‘2022-02-20’, ‘orange and white’, 5.5)]

Utilizar bases de datos en memoria tiene algunas desventajas. Si un programa falla por una excepción sin manejar, perderemos la base de datos. Podemos mitigar el riesgo poniendo el código en una declaración try que capture cualquier excepción no manejada y utilice una declaración except para guardar el archivo en una base de datos. El capítulo 4 habla del manejo de excepciones con las declaraciones try y except.

Copiar bases de datos

Podemos obtener una copia de una base de datos llamando al método iterdump() con el objeto Connection. Este método devuelve un iterador que genera el texto de las instrucciones SQLite necesarias para recrear la base de datos. Podemos usar los iteradores en bucles for o pasarlos a la función list() para convertirlos en una lista de cadenas. Por ejemplo, para obtener las instrucciones SQLite necesarias para recrear la base de datos sweigartcats.db, introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > import sqlite3

> > > conn = sqlite3.connect('sweigartcats.db',
isolation_level=None)

> > > with open('sweigartcats-queries.txt', 'w',
encoding='utf-8') as fileObj:

... for line in conn.iterdump():

. fileObj.write(line + '\n')
```

Este código crea un archivo sweigartcats-queries.txt con las siguientes instrucciones SQLite, que pueden recrear la base de datos:

```
BEGIN TRANSACTION;
```

```
CREATE TABLE "cats" (name TEXT NOT NULL, birthdate TEXT, fur
```

```
TEXT, weight_kg REAL) STRICT;
```

```
INSERT INTO "cats" VALUES('Zophie','2021-01-24','gray  
tabby',5.6);
```

```
INSERT INTO "cats" VALUES('Miguel','2016-12-24','siamese',6.2);
```

```
INSERT INTO "cats" VALUES('Jacob','2022-02-20','orange and  
white',5.5);
```

--fragmento omitido--

```
INSERT INTO "cats" VALUES('Spunky','2015-09-04','gray',5.9);
```

```
INSERT INTO "cats" VALUES('Shadow','2021-01-18','calico',6.0);
```

```
COMMIT;
```

El texto de estas instrucciones será seguramente más grande que la base de datos original, pero las instrucciones tienen la ventaja de ser legibles para humanos y fáciles de editar antes de copiar y pegar en el código Python o en una *app* con SQLite, como veremos a continuación.

Apps con SQLite

A veces, es posible que quieras investigar una base de datos SQLite directamente sin tener que escribir código Python superfluo. Puedes hacerlo instalando el comando `sqlite3`, que se ejecuta desde una ventana de terminal de línea de comandos y está documentado en <https://sqlite.org/cli.html>.

En Windows, descarga los archivos con la descripción “A bundle of command line tools for managing SQLite database files” de <https://www2.sqlite.org/matrix/download.html> y pon el programa `sqlite3.exe` en una carpeta en la variable `PATH` del sistema. (Consulta el capítulo 12 para más información sobre la variable de entorno `PATH` y las ventanas de terminal). El comando `sqlite3` está preinstalado en macOS. Para UbuntuLinux, ejecuta `sudo apt install sqlite3` para instalarlo.

A continuación, en una ventana de terminal, ejecuta `sqlite3 example.db` para conectar con la base de datos de `example.db`. Si este archivo no existe, `sqlite3` lo crea con una base de datos vacía. Puedes introducir instrucciones SQL en esta herramienta, aunque, a diferencia de las que pasamos a `conn.execute()` en Python, deben acabar con punto y coma. Por ejemplo, escribe lo siguiente en la ventana de terminal:

```
C:\Users\Al> sqlite3 example.db
```

```
SQLite version 3.xx.xx
```

```
Enter “.help” for usage hints.
```

```
sqlite> CREATE TABLE IF NOT EXISTS cats (name TEXT NOT NULL,
```

```
birthdate TEXT, fur TEXT, weight_kg REAL) STRICT;
```

```
sqlite> INSERT INTO cats VALUES ('Zophie', '2021-01-24', 'gray tabby', 4.7);
```

```
sqlite> SELECT * from cats;
```

```
Zophie|2021-01-24|gray tabby|4.7
```

Como verás en este ejemplo, la herramienta de línea de comandos `sqlite3` proporciona una especie de consola interactiva de SQLite para introducir instrucciones en su *prompt* `sqlite>`. El comando `.help` muestra comandos adicionales, como `.tables` (que muestra las tablas de la base de datos) y `.headers` (que permite activar y desactivar encabezados de columna):

```
sqlite> .tables
```

```
cats
```

```
sqlite> .headers on
```

```
sqlite> SELECT * from cats;
```

```
name|birthdate|fur|weight_kg
```

Si la herramienta de línea de comandos te resulta demasiado dispersa, también hay *apps* gratuitas de código abierto para ver bases de datos SQLite con una interfaz gráfica de usuario (GUI) en Windows, macOS y Linux:

- DB Browser for SQLite (<https://sqlitebrowser.org>).
- SQLite Studio (<https://sqlitestudio.pl>).
- DBeaver Community (<https://dbeaver.io>).

Aunque estas *apps* con GUI facilitan el trabajo con bases de datos SQLite, sigue mereciendo la pena aprender la sintaxis basada en texto de las consultas e instrucciones de SQLite.

Resumen

Los ordenadores hacen posible tratar con grandes cantidades de datos, pero simplemente meter datos en un archivo de texto o una hoja de cálculo puede no organizarlos lo suficientemente bien como para utilizarlos con eficiencia. Las bases de datos SQL como SQLite ofrecen una manera avanzada de almacenar grandes cantidades de información y también de recuperar los datos concretos que queramos con el lenguaje SQL.

SQLite es una base de datos impresionante y Python incluye el módulo `sqlite3` en su librería estándar. La versión de SQL de SQLite es diferente de la utilizada en otras bases de datos relacionales, pero lo bastante parecida como para que SQLite suponga una buena introducción a las bases de datos en general.

Las bases de datos SQLite se encuentran en un único archivo sin un servidor específico. Pueden contener varias tablas (que puedes imaginar como hojas de cálculo), y cada tabla puede tener varias columnas. Para editar (crear, leer, actualizar y borrar) los valores de una tabla, puedes usar operaciones CRUD con las instrucciones `INSERT`, `SELECT`, `UPDATE` y `DELETE`. Para cambiar las propias tablas y columnas, puedes usar las instrucciones `ALTER TABLE` y `DROP TABLE`. Por último, las claves foráneas permiten vincular

registros en varias tablas con una técnica llamada unión interna.

Hay mucho más sobre SQLite y las bases de datos de lo que puede abarcar un capítulo. Si deseas saber más sobre las bases de datos SQL en general, te recomiendo *Practical SQL, 2nd edition* (No Starch Press, 2022), de Anthony DeBarros.

Preguntas prácticas

1. ¿Qué instrucciones de Python obtendrán un objeto Connection para una base de datos SQLite en un archivo llamado example.db?
2. ¿Qué instrucción de Python creará una tabla nueva llamada students con columnas TEXT llamadas first_name, last_name y favorite_color?
3. ¿Cómo conectamos con una base de datos SQLite en modo autocommit?
4. ¿Cuál es la diferencia entre los tipos de datos INTEGER y REAL en SQLite?
5. ¿Qué añade a una tabla el modo estricto?
6. ¿Qué significa * en la consulta 'SELECT * FROM cats'?
7. ¿Qué significa CRUD?
8. ¿Qué significa ACID?
9. ¿Qué instrucción añade registros a una tabla?
10. ¿Qué instrucción elimina registros de una tabla?
11. ¿Qué pasa si no especificamos la proposición WHERE en una instrucción UPDATE?
12. ¿Qué es un índice? ¿Qué código crearía un índice para una columna llamada birthdate en una tabla llamada cats?
13. ¿Para qué sirve una clave foránea?

14.¿Cómo podemos borrar una tabla llamada cats?

15.¿Qué “nombre de archivo” especificarías para crear una base de datos en memoria?

16.¿Cómo se copia una base de datos en otra?

Programas de práctica

Para practicar, escribe programas que hagan las siguientes tareas.

Comprobador de vacunación de gatos

Descarga la base de datos sweigartcats.db de la página de recursos del libro. Escribe un programa que la abra y haga una lista con todos los que no tengan las vacunas llamadas ‘rabies’, ‘FeLV’, y ‘FVRCP’. Comprueba también la base de datos por si hay errores buscando todas las vacunas puestas en una fecha anterior a la fecha de nacimiento del gato.

Base de datos de ingredientes de comidas

Escribe un programa que cree dos tablas, una para comidas (meals) y otra para ingredientes (ingredients), utilizando estas instrucciones SQL:

```
CREATE TABLE IF NOT EXISTS meals (name TEXT) STRICT
```

```
CREATE TABLE IF NOT EXISTS ingredients (name TEXT,
```

```
meal_id INTEGER, FOREIGN KEY(meal_id) REFERENCES meals
```

```
(rowid)) STRICT
```

Después, escribe un programa que pida entrada de usuario. Si el usuario introduce ‘quit’, el programa debería terminar. El usuario también puede introducir una nueva comida, seguida de punto y coma, y una lista de ingredientes separados por comas: ‘meal:ingredient1,ingredient2’. Guarda la comida y sus ingredientes en las tablas meals e ingredients.

Por último, el usuario puede introducir el nombre de una comida o ingrediente. Si está en la tabla meals, el programa debería enumerar los ingredientes correspondientes. Si el nombre está en la tabla ingredients, el programa debería enumerar todas las comidas que utilicen este ingrediente. Por ejemplo, la salida del programa podría ser así:

> **onigiri:rice,nori,salt,sesame seeds**

Meal added: onigiri

> **chicken and rice:chicken,rice,cream of chicken soup**

Meal added: chicken and rice

> **onigiri**

Ingredients of onigiri:

rice

nori

salt

sesame seeds

> **chicken**

Meals that use chicken:

chicken and rice

> **rice**

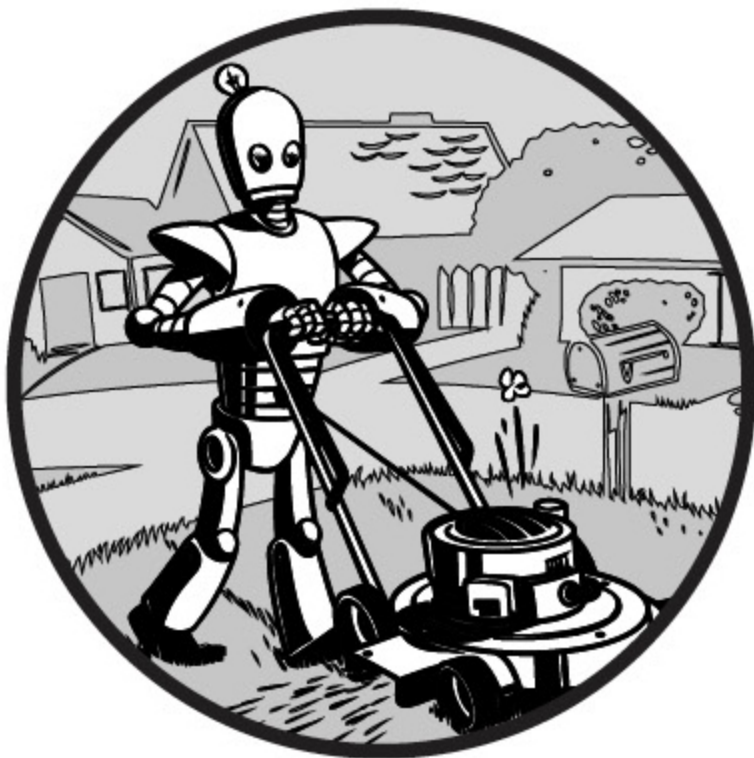
Meals that use rice:

onigiri

chicken and rice

> **quit**

Documentos de Word y PDF



Aunque podrías pensar en PDF y Word como formatos para almacenar texto, estos documentos son archivos binarios que también contienen información de fuentes, colores y disposición, lo que hace que sean mucho más complejos que los archivos de texto simple. Si quieres que tus programas lean o escriban documentos en PDF o Word, tendrás que hacer algo más que simplemente pasar el nombre de archivo a `open()`. Por suerte, hay muchos paquetes de Python que facilitan estas interacciones. Este capítulo hablará de dos de ellos: PyPDF y Python-Docx.

Documentos en PDF

PDF significa *Portable Document Format*, formato de documento portátil, y utiliza la extensión .pdf. Aunque los PDF son compatibles con muchas funciones, este apartado se centrará en tres tareas habituales: extraer el contenido textual de un documento, extraer sus imágenes y crear nuevos PDF a partir de documentos existentes.

PyPDF es un paquete de Python para crear y modificar archivos PDF. Instálalo siguiendo las instrucciones del apéndice A. Si el paquete se instaló correctamente, ejecutar `import pypdf` en el intérprete de comandos interactivo no debería dar ningún error.

Aunque los PDF son geniales para organizar el texto de forma que resulte a la gente fácil leer o imprimir, no son fáciles de analizar en texto simple. En consecuencia, PyPDF puede cometer errores al extraer texto de un PDF e incluso puede tener problemas para abrir algunos archivos. Por desgracia, no hay mucho que podamos hacer al respecto. Es posible que PyPDF sea incapaz de trabajar con algunos de tus archivos. Dicho esto, yo todavía no he encontrado un archivo PDF que no pudiese abrir PyPDF.

Extraer texto

Para empezar a trabajar con PyPDF, vamos a usar el PDF de un capítulo de muestra de mi libro sobre algoritmos recursivos, *The Recursive Book of Recursion* (No Starch Press, 2022), que se muestra en la figura 17.1.

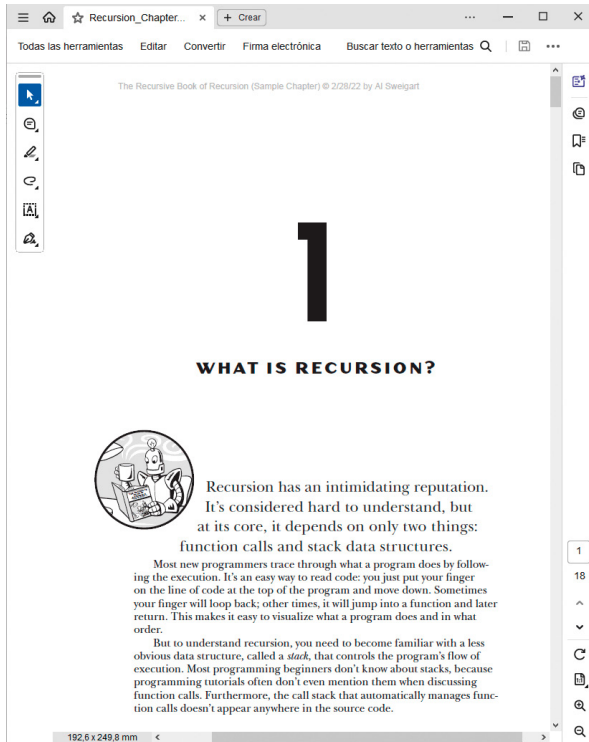


Figura 17.1. El archivo PDF del que extraeremos texto.

Descarga el archivo `Recursion_Chapter1.pdf` de la página de recursos en línea del libro e introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > import pypdf  
1 > > > reader = pypdf.PdfReader('Recursion_Chapter1.pdf')  
2 > > > len(reader.pages)  
18
```

Importa el módulo `pypdf` y llama a `pypdf.PdfReader()` con el nombre del PDF para obtener un objeto `PdfReader` que representa al PDF 1. Guarda este objeto en una variable llamada `reader`. El atributo `pages` del objeto `PdfReader` es una estructura de datos de estilo lista de objetos `Pages`, que representan páginas individuales

del PDF. Igual que sucede con las listas de Python, podemos pasar esta estructura de datos a la función `len()` 2. Este PDF de ejemplo tiene 18 páginas.

Para extraer el texto de este PDF y ponerlo en un archivo de texto, abre una nueva pestaña en el editor de archivos y guarda el siguiente código como `extractpdftext.py`:

```
import pypdf

import pdfminer.high_level

PDF_FILENAME = 'Recursion_Chapter1.pdf'

TEXT_FILENAME = 'recursion.txt'

text = ''

try:

    reader = pypdf.PdfReader(PDF_FILENAME)

    1 for page in reader.pages:

    2 text += page.extract_text()

except Exception:

    3 text = pdfminer.high_level.extract_text(PDF_FILENAME)

with open(TEXT_FILENAME, 'w', encoding='utf-8') as file_obj:

    4 file_obj.write(text)
```

Usamos el módulo `pypdf` para extraer el texto, pero, si falla con un PDF en particular y lanza una excepción, recurriremos al módulo `pdfminer`. Dentro de un bucle `try`, usamos un bucle `for` 1 para iterar por cada objeto `Page` del objeto `PdfReader` del archivo PDF. Llamar al método `extract_text()` del objeto `Page` 2 devuelve una cadena que podemos concatenar con la siguiente variable de texto. Cuando el bucle termine, `text` contendrá una única cadena con todo el texto del PDF.

Si el PDF tiene un formato poco convencional que no entiende PyPDF, podemos probar a usar `pdfminer.high_level`, un módulo más antiguo incluido en los paquetes de terceros de este libro. La función `extract_text()` del módulo contiene el contenido del PDF como una única cadena en lugar de trabajar con las páginas de una en una 3.

Por último, podemos usar la función `open()` y el método `write()`, explicados en el capítulo 10, para escribir la cadena en un archivo de texto 4.

Postprocesamiento con IA

La extracción de texto que acabamos de hacer no es perfecta. El formato de archivo PDF es infame por su enrevesamiento y se diseñó originalmente para imprimir documentos, no para que los leyeren máquinas. Incluso aunque no haya problemas con la extracción, la disposición del texto es fija: la cadena contendrá saltos de línea después de cada fila de texto y palabra con guion al final de una fila. Por ejemplo, el texto extraído de nuestro PDF de ejemplo queda así:

1

WHAT IS RECURSION?

Recursion has an intimidating reputation.

It's considered hard to understand, but

at its core, it depends on only two things:

function calls and stack data structures.

Most new programmers trace through what a program does by follow -

ing the execution. It's an easy way to read code: you just put your finger

--*fragmento omitido*--

Como verás, hay varias decisiones subjetivas que tomar:

- ¿Dónde deberían empezar y acabar los párrafos del PDF?
- ¿Deberían los números de página, encabezados y pies de página incluirse en el texto extraído?
- ¿Cómo deberían convertirse a texto simple las tablas de datos del PDF?
- ¿Cuánto espacio en blanco debería incluir el texto extraído?

Limpiar este texto es aburrido y no es fácil automatizar la tarea con código. Sin embargo, un modelo de lenguaje de gran tamaño (LLM) como ChatGPT puede entender el contexto del texto lo bastante bien como para producir una versión limpia automáticamente. Usa un *prompt* como el siguiente antes de copiar y pegar el texto extraído:

Lo siguiente es un texto extraído de varias páginas de un PDF de un libro sobre algoritmos recursivos. Limpia el texto. Con esto quiero decir que pongas los párrafos en línea aparte. Quita además el encabezado y el pie de cada página y deshazte de los guiones al final de cada línea para las palabras divididas. No hagas correcciones de ortografía o gramática ni cambies la redacción. Aquí está el texto...

En una prueba, este *prompt* produjo el siguiente texto:

WHAT IS RECURSION?

Recursion has an intimidating reputation. It's considered hard to understand, but at its core, it depends on only two things: function calls and stack data structures. Most new programmers trace through what a program does by following the execution. It's an easy way to read code: you just put your finger...

Un humano debe revisar siempre la salida de cualquier sistema de

IA. Por ejemplo, el LLM ha quitado el número 1 del capítulo del principio del texto, y no era esa mi intención. Puede que haya que ajustar el *prompt* para corregir cualquier malentendido.

Si no tienes acceso a un LLM, la documentación de PyPDF ofrece una lista de trucos de postprocesamiento con *snippets* de código en <https://pypdf.readthedocs.io/en/latest/user/post-processing-in-text-extraction.html>.

Extraer imágenes

PyPDF también puede extraer las imágenes de un documento en PDF. Cada objeto Page tiene un atributo `images` que contiene una estructura de datos tipo lista con objetos `Image`. Podemos escribir los bytes de estos objetos `Image` en un archivo de imagen abierto en modo de escritura binaria, `'wb'`. Un objeto `Image` también tiene un atributo `name` que contiene una cadena con el nombre de la imagen. Aquí tenemos código que extrae las imágenes de todas las páginas del PDF del capítulo de muestra. Abre una pestaña nueva en el editor de archivos y guarda este código como `extractpdfimages.py`:

```
import pypdf

PDF_FILENAME = 'Recursion_Chapter1.pdf'

reader = pypdf.PdfReader(PDF_FILENAME)

1 image_num = 0

2 for i, page in enumerate(reader.pages):

    print(f'Reading page {i + 1} - {len(page.images)} images found...')

    try:

        3 for image in page.images:

            4 with open(f'{image_num}_page{i + 1}_{image.name}', 'wb') as file:

                5 file.write(image.data)
```

```
print(f'Wrote {image_num}_page{i + 1}_{image.name}...')
```

```
6 image_num += 1
```

except Exception as exc:

```
7 print(f'Skipped page {i + 1} due to error: {exc}')
```

La salida de este programa será así:

Reading page 1 - 7 images found...

Wrote 0_page1_Im0.jpg...

Wrote 1_page1_Im1.png...

--*fragmento omitido*--

Reading page 7 - 1 images found...

Skipped page 7 due to error: not enough image data

--*fragmento omitido*--

Reading page 17 - 0 images found...

Reading page 18 - 0 images found...

Las imágenes de un PDF suelen tener nombres genéricos, como Im0.jpg o Im1.png, así que usamos un contador de variables llamado image_num 1 junto con el número de página para asignarles nombres únicos. Primero, pasamos en bucle por cada objeto Page del atributo pages del objeto PdfReader. Recuerda que la función enumerate() de Python 2 devuelve índices enteros y el elemento de lista del objeto de tipo lista que le pasemos. Cada objeto Page tiene un atributo images por el que también iteraremos 3. Dentro de ese segundo bucle for anidado que itera por los objetos Image del atributo images, llamamos a open() y usamos una cadena formateada para obtener un nombre de archivo 4. Este nombre de archivo se compone del entero del contador image_num, el número de imagen y la cadena del atributo name del objeto Image. Como i empieza en 0 y los números de página del PDF en 1, usaremos i + 1

para guardar el número de página. El nombre incluirá también la extensión de archivo, como .png o .jpg. También debemos pasar 'wb' en la llamada a la función open() para que el archivo se abra en modo escritura binaria. Los bytes del archivo de imagen se almacenan en el atributo data del objeto Image, que pasamos al método write() 5. Una vez escrita una imagen, el código incrementa image_num en 1 6.

Si alguna incompatibilidad entre el archivo PDF y PyPDF hace que el atributo images de un objeto Page lance una excepción, nuestras declaraciones try y except pueden capturarla e imprimir un breve mensaje de error 7. De ese modo, un problema con una página no hará que falle todo el programa. Al igual que la extracción de texto, la extracción de imágenes puede no ser perfecta. Por ejemplo, PyPDF no detectó varias de las imágenes del PDF de ejemplo y mostró un mensaje de error en su lugar. Sin embargo, es sorprendente que PyPDF extraiga pequeñas imágenes en blanco usadas como fondo o separadores. Al trabajar con PDF, necesitarás a menudo una revisión humana para asegurarte de que la salida es aceptable.

Crear PDF a partir de otras páginas

El complementario de PdfReader en PyPDF es PdfWriter, que puede crear nuevos archivos PDF. Pero PyPDF no puede escribir un texto cualquiera en PDF como hace Python con los archivos de texto simple. Las capacidades de escribir PDF de PyPDF se limitan a copiar, fusionar, recortar y transformar páginas de otros PDF en archivos nuevos. El código de este ejemplo para el intérprete de comandos interactivo crea una copia del capítulo de muestra con solo las cinco primeras páginas:

```
> > > import pypdf

1 > > > writer = pypdf.PdfWriter()

2 > > > writer.append('Recursion_Chapter1.pdf', (0, 5))

> > > with open('first_five_pages.pdf', 'wb') as file:

3 . writer.write(file)
```

(False, <_io.BufferedWriter name = 'first_five_pages.pdf' >)

Primero, creamos un objeto PdfWriter llamando a `pypdf.PdfWriter()`

1. El objeto PdfWriter de la variable `writer` representa un PDF en blanco con cero páginas. A continuación, el método `append()` del objeto PdfWriter copia las cinco primeras páginas del PDF del capítulo de ejemplo, que identifica con el nombre de archivo 'Recursion_Chapter1.pdf' 2. (Pese a que el nombre es idéntico, el método `append()` del objeto PdfWriter no es igual que el método de lista `append()`).

El segundo argumento para este método es la tupla (0, 5), que dice al objeto PdfWriter que copie páginas empezando desde el índice de página 0 (la primera página del objeto PdfWriter) hasta, pero sin incluir, el índice de página 5. PyPDF considera que el índice 0 es la primera página, incluso aunque las aplicaciones de PDF la llamen página 1.

Por último, para escribir el contenido del objeto PdfWriter en un archivo PDF, llama a `open()` con el nombre de archivo y el modo 'wb', y pasa el objeto File al método `write()` del objeto PdfWriter 3. Esto debería generar un nuevo archivo PDF.

La tupla proporcionada para `append()` puede contener dos o tres enteros. Si se proporciona un tercer entero, el método se salta ese número de páginas. Como este comportamiento coincide con la función `range()`, podríamos pasar los dos o tres enteros a `list(range())` para ver qué páginas copiaría el código:

```
> > > list(range(0, 5)) # Pasar (0, 5) hace que append() copie estas páginas:
```

```
[0, 1, 2, 3, 4]
```

```
> > > list(range(0, 5, 2)) # Pasar (0, 5, 2) hace que append() copie estas páginas:
```

```
[0, 2, 4]
```

El método `append()` también puede aceptar un argumento de lista

con enteros de número de página para cada página que añadir. Por ejemplo, supongamos que reemplazamos el código del ejemplo anterior por este:

```
> > > writer.append('Recursion_Chapter1.pdf', [0, 1, 2, 3, 4])
```

Este código también copiará las cinco primeras páginas del PDF en el objeto PdfWriter. Observa que append() interpreta de manera diferente los argumentos de tupla y de lista; la tupla (0, 5) le dice que copie las páginas del índice 0 hasta el 5, pero sin incluirlo, y la lista [0, 5] le dice que copie individualmente la página del índice 0 y, después, la del 5. Esta diferencia entre las tuplas y las listas no es convencional y no la encontrarás en otras librerías de Python, pero es parte del diseño de PyPDF.

El método append() añade las páginas copiadas al final del objeto PdfWriter. Para insertar las páginas copiadas antes del final, hay que llamar al método merge(), que tiene un argumento entero adicional que especifica dónde insertar las páginas. Por ejemplo, fíjate en este código:

```
> > > writer.merge(2, 'Recursion_Chapter1.pdf', (0, 5))
```

Este código copia las páginas desde el índice 0 hasta, sin incluir, el 5 y las inserta donde está el índice de página 2 (la tercera página) en el objeto PdfWriter de writer. La página que estaba originalmente en el índice 2 y las siguientes se mueven con la inserción de páginas.

Rotar páginas

También podemos rotar las páginas de un PDF en aumentos de 90 grados con el método rotate() del objeto Page. Pasa 90, 180 o 270 como argumento a este método para rotar la página en el sentido de las agujas del reloj, y -90, -180 o -270 para girarlas en sentido contrario. Rotar páginas es útil cuando se tienen varios PDF que, por alguna razón, no están bien puestos y hay que girarlos, o cuando se necesita rotar solo unas pocas páginas en un archivo PDF. Las aplicaciones de PDF suelen incluir funciones de rotación que podemos usar para corregir manualmente los PDF, pero Python permite aplicar rápidamente rotaciones a varios PDF para

automatizar esta tediosa tarea.

Por ejemplo, escribe lo siguiente en el intérprete de comandos interactivo para rotar las páginas del PDF de muestra:

```
> > > import pypdf

> > > writer = pypdf.PdfWriter()

> > > writer.append('Recursion_Chapter1.pdf')

1 > > > for i in range(len(writer.pages)):

. 2 writer.pages[i].rotate(90)

.

{'/ArtBox': [21, 21, 525, 687], '/BleedBox': [12, 12, 534, 696],

--fragmento omitido--

> > > with open('rotated.pdf', 'wb') as file:

... writer.write(file)

.

(False, <_io.BufferedWriter name='rotated.pdf'>)
```

Podemos crear un nuevo objeto PdfWriter y copiar las páginas del capítulo de muestra ahí. Después, usamos un bucle for para pasar por cada número de página. La llamada a len(writer.pages) devuelve el número de páginas 1 como un entero. La expresión writer.pages[i] accede a cada objeto Page en una iteración del bucle for, y la llamada al método rotate(90) 2 rota esta página en el objeto PdfWriter.

El PDF resultante debería tener todas las páginas rotadas 90 grados en el sentido de las agujas del reloj, como muestra la figura 17.2. PyPDF no puede rotar documentos en incrementos distintos de 90 grados.

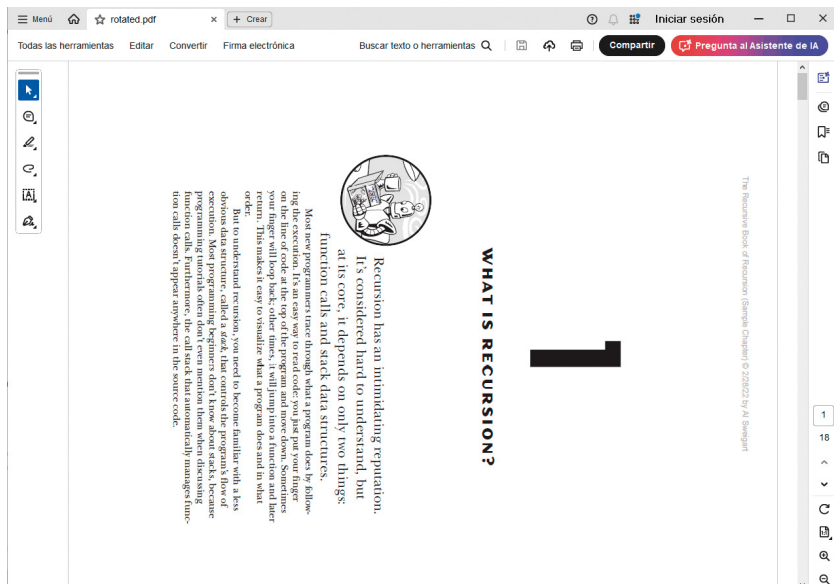


Figura 17.2. El archivo rotated.pdf con la página girada 90 grados en el sentido de las agujas del reloj.

Insertar páginas en blanco

Podemos insertar o añadir una página en blanco a un objeto PdfWriter con los métodos `insert_blank_page()` y `add_blank_page()`. El tamaño de la página nueva será el mismo que el de la página anterior. Por ejemplo, vamos a crear una copia del PDF del capítulo de muestra con páginas en blanco al final y en la página 3:

```
> > import pypdf
> > writer = pypdf.PdfWriter()
> > writer.append('Recursion_Chapter1.pdf')
1 > > writer.add_blank_page()
{'/Type': '/Page', '/Resources': {}, '/MediaBox': [0.0, 0.0,
546, 708], '/Parent': IndirectObject(1, 0, 2629126028624)}
```



```
2 >>> writer.insert_blank_page(index = 2)
```

```
{‘/Type’: ‘/Page’, ‘/Parent’: NullObject, ‘/Resources’: {},  
‘/MediaBox’: RectangleObject([0.0, 0.0, 546, 708])}
```

```
>>> with open(‘with_blanks.pdf’, ‘wb’) as file:
```

```
... writer.write(file) # Guarda el objeto writer en un archivo PDF.
```

```
.
```

```
(False, <_io.BufferedWriter name = ‘with_blanks.pdf’>)
```

Una vez copiadas todas las páginas del PDF de ejemplo en el objeto PdfWriter, el método `add_blank_page()` añade una página en blanco al final del documento. El método `insert_blank_page()` inserta una página en blanco en el índice 2 (que es la tercera página porque la primera está en el índice 0). Este método requiere especificar el nombre del parámetro `index`.

Podemos dejar estas páginas en blanco o añadirles contenido después, por ejemplo, superposiciones y marcas de agua, como veremos en el próximo apartado.

Añadir marcas de agua y superposiciones

PyPDF también puede superponer el contenido de una página encima de otra, lo cual es útil para añadir un logotipo, un sello de tiempo o una marca de agua a una página. En PyPDF, un sello o superposición es contenido colocado encima del contenido existente en una página, mientras que una marca de agua es contenido colocado debajo del contenido existente en la página.

Descarga `watermark.pdf` de los recursos en línea del libro y ponlo en el directorio de trabajo actual, junto con el PDF del capítulo de muestra. A continuación, introduce esto en el intérprete de comandos interactivo:

```
>>> import pypdf
```

```
>>> writer = pypdf.PdfWriter()
```

```

> > > writer.append('Recursion_Chapter1.pdf')

1 > > > watermark_page =
pypdf.PdfReader('watermark.pdf').pages[0]

> > > for page in writer.pages:

2. page.merge_page(watermark_page, over = False)

.

> > > with open('with_watermark.pdf', 'wb') as file:

... writer.write(file)

.

(False, <_io.BufferedWriter name='with_watermark.pdf'>)

```

Este ejemplo crea una copia del PDF del capítulo de muestra en un nuevo objeto PdfWriter, guardado en la variable writer. También obtiene el objeto Page para la primera página del PDF de la marca de agua y lo guarda en la variable watermark_page. Después, el bucle for pasa por todos los objetos Page del objeto PdfWriter y aplica la marca de agua pasándosela a merge_page(). (No confundas el método merge_page() de los objetos Page con el método merge() de los objetos PdfWriter que hemos visto anteriormente).

El método merge_page() también tiene un argumento over al que pasaremos True para crear un sello o superposición o False para crear una marca de agua.

Una vez modificadas las páginas del objeto PdfWriter en el bucle, el código lo guarda como with_watermark.pdf. La figura 17.3 muestra el PDF original de la marca de agua y dos páginas del capítulo de muestra con la marca de agua aplicada.

Top secret
Do not accidentally email to wikileaks

Menú

with_watermark.pdf

+ Crear

Iniciar sesión

Todas las herramientasEditarConvertirFirma electrónicaBuscar texto o herramientas

The Recursive Book of Recursion (Sample Chapter) © 2016/22 by Al Sweigart

Top secret

Do not accidentally email to wikileaks

WHAT IS RECURSION?



Recursion has an intimidating reputation. It's considered hard to understand, but at its core, it depends on only two things: function calls and stack data structures.

Most new programmers trace through what a program does by following the execution. It's an easy way to read code: you just put your finger on the line of code at the top of the program and move down. Sometimes your finger will loop back; other times, it will jump into a function and later return. This makes it easy to visualize what a program does and in what order.

But to understand recursion, you need to become familiar with a less obvious data structure, called a *stack*, that controls the program's flow of execution. Most programming beginners don't know about stacks, because programming tutorials often don't even mention them when discussing function calls. Furthermore, the call stack that automatically manages function calls doesn't appear anywhere in the source code.

1

18

Navigation icons: back, forward, search, etc.

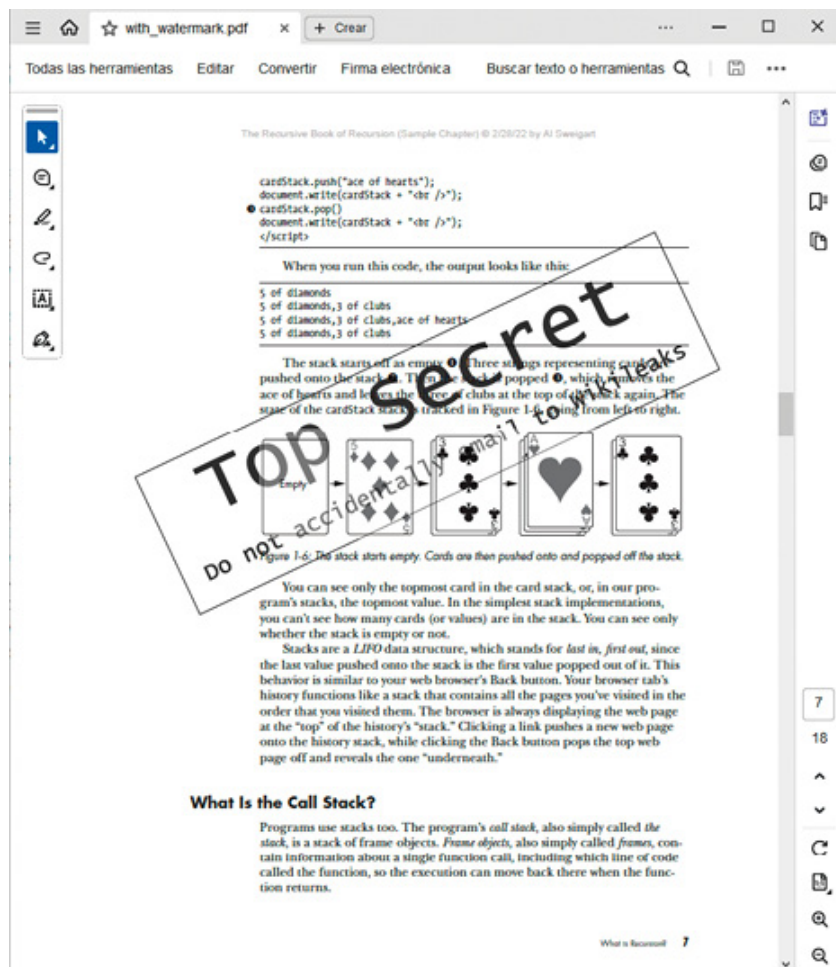


Figura 17.3. El PDF de la marca de agua (arriba) y páginas con la marca añadida (centro, abajo).

El método `merge_page()` es útil para hacer grandes cambios en documentos en PDF, como combinar el contenido de dos páginas.

Encriptar y desencriptar archivos PDF

Los PDF permiten encriptar su contenido, lo que los hace ilegibles. La encriptación solo tiene la fuerza de la contraseña que elijas, así que crea una contraseña que use distintos tipos de caracteres, no sea una palabra del diccionario y tenga entre 14 y 16 caracteres.

Ten en cuenta que los PDF no cuentan con mecanismos para restablecer contraseñas, si la olvidas, el archivo quedará ilegible para siempre a menos que consigas averiguarla.

El método `encrypt()` de los objetos `PdfWriter` acepta una cadena de contraseña y una cadena que selecciona el algoritmo de encriptación. El argumento `'AES-256'` implementa el algoritmo de encriptación moderno recomendado, así que siempre usaremos ese. Introduce lo siguiente en el intérprete de comandos interactivo para crear una copia encriptada del PDF del capítulo de muestra:

```
> > > import pypdf
> > > writer = pypdf.PdfWriter()
> > > writer.append('Recursion_Chapter1.pdf')
> > > writer.encrypt('swordfish', algorithm='AES-256')
> > > with open('encrypted.pdf', 'wb') as file:
... writer.write(file)
.
```

```
(False, <_io.BufferedWriter name='encrypted.pdf'>)
```

La llamada al método `encrypt('swordfish', algorithm='AES-256')` con el objeto `PdfWriter` encripta el contenido del PDF. Una vez escrito este PDF encriptado en el archivo `encrypted.pdf`, ninguna aplicación de PDF, incluido PyPDF, debería ser capaz de abrirlo sin introducir la contraseña `swordfish`. (Es una mala contraseña porque es una palabra que existe y, por tanto, es fácil de adivinar). Los datos encriptados parecen aleatorios hasta que les aplicas la clave de desencriptación correcta; desencriptar el documento con la contraseña incorrecta produce datos inservibles. Las aplicaciones de PDF lo detectarán y te pedirán la contraseña de nuevo.

PyPDF puede aplicar una contraseña a un PDF encriptado para desencriptarlo. Introduce lo siguiente en el intérprete de comandos interactivo para detectar PDF encriptados con el atributo `is_encrypted` y desencriptarlos con `decrypt()`:

```
> > > import pypdf
```

```
1 > > > reader = pypdf.PdfReader('encrypted.pdf')
```

```
> > > writer = pypdf.PdfWriter()
```

```
2 > > > reader.is_encrypted
```

```
True
```

```
3 > > > reader.pages[0]
```

```
Traceback (most recent call last):
```

```
--fragmento omitido--
```

```
pypdf.errors.FileNotDecryptedError: File has not been decrypted
```

```
4 > > > reader.decrypt('an incorrect password').name
```

```
'NOT_DECRYPTED'
```

```
5 > > > reader.decrypt('swordfish').name
```

```
'OWNER_PASSWORD'
```

```
6 > > > writer.append(reader)
```

```
> > > with open('decrypted.pdf', 'wb') as file:
```

```
... writer.write(file)
```

```
.
```

```
(False, <_io.BufferedWriter name='decrypted.pdf'>)
```

Cargamos el PDF encriptado en un objeto PdfReader como cualquier otro PDF 1. El objeto PdfReader tiene un atributo is_encrypted 2 establecido en True o False. Si intentas leer el contenido del PDF accediendo, por ejemplo, al atributo pages 3, PyPDF lanza un FileNotDecryptedError porque es incapaz de leerlo.

Los PDF pueden tener una contraseña de usuario que permita ver el

PDF y una contraseña de propietario que te permita dar permisos para imprimir, comentar, extraer texto y otras funciones. La contraseña de usuario y la contraseña de propietario son el primer y el segundo argumento de `encrypt()`, respectivamente. Si solo se pasa un argumento de cadena a `encrypt()`, PyPDF lo usará para ambas contraseñas.

Para descryptar el objeto `PdfReader`, llama al método `decrypt()` y pásale la cadena de la contraseña. Este método devuelve un objeto `PasswordType`; solo nos interesa el atributo `name` de este objeto. Si `name` está establecido en `'NOT_DECRYPTED'` 4, hemos dado la contraseña incorrecta. Si está establecido en `'OWNER_PASSWORD'` o `'USER_PASSWORD'` 5, hemos introducido bien la contraseña de usuario o propietario.

Ahora podemos añadir las páginas del objeto `PdfReader` a un objeto `PdfWriter` 6 y guardar el PDF descryptado en un archivo.

Proyecto 12: Combinar páginas seleccionadas de varios PDF

Supongamos que se te ha encomendado la tediosa tarea de fusionar decenas de documentos en PDF en un único archivo PDF. La primera página de cada documento es una portada, pero no queremos que estén repetidas en el resultado final. Aunque hay muchos programas gratuitos para combinar PDF, la mayoría de ellos solo fusionan los archivos completos. Vamos a escribir un programa de Python para personalizar las páginas que queremos incluir en el PDF combinado.

A alto nivel, esto es lo que hará el programa:

- Buscar todos los archivos PDF en el directorio de trabajo actual y ordenarlos alfabéticamente.
- Para cada PDF, copiar todas las páginas después de la primera en un PDF de salida.
- Guardar el PDF de salida en un archivo.

En términos de implementación, el código tendrá que hacer lo siguiente:

- Llamar a `os.listdir()` para encontrar todos los archivos del directorio de trabajo actual y quitar todos los que no sean PDF. (Vimos esta función en el capítulo 11).
- Llamar al método de lista de Python `sort()` para ordenar los nombres de archivo alfabéticamente.
- Crear un objeto `PdfWriter` para el PDF de salida.
- Pasar en bucle por cada archivo PDF, creando un objeto `PdfReader` para él.
- Desde el objeto `PdfReader`, copiar en el PDF de salida todas las páginas después de la primera.
- Escribir el PDF de salida en un archivo.

Abre una pestaña nueva para este proyecto en el editor de archivos y guárdalo como `combine_pdfs.py`.

Paso 1: buscar todos los archivos PDF

Primero, el programa necesita obtener una lista de todos los archivos con la extensión `.pdf` en el directorio de trabajo actual y ordenarlos. Escribe lo siguiente en el programa:

```
# combine_pdfs.py - Combina todos los PDF del directorio de
trabajo actual
```

```
# en un único archivo PDF
```

```
1 import pypdf, os
```

```
# Obtiene todos los nombres de los archivos PDF.
```

```
pdf_filenames = []
```

```
for filename in os.listdir('.'):

```

```
    if filename.endswith('.pdf'):

```

```
        pdf_files.append(filename)
```

```
3 pdf_filenames.sort(key = str.lower)
```

```
4 writer = pypdf.PdfWriter()
```

```
# PENDIENTE: Pasar en bucle por todos los archivos PDF.
```

```
# PENDIENTE: Copiar todas las páginas después de la primera.
```

```
# PENDIENTE: Guardar el PDF resultante en un archivo.
```

Este código importa los módulos pypdf y os 1. La llamada a os.listdir('.') devuelve una lista de todos los archivos del directorio de trabajo actual. El código pasa entonces en bucle por esa lista, añadiendo archivos con la extensión .pdf a una lista en la variable pdf_filenames 2. A continuación, ordenamos esta lista alfabéticamente con el argumento key = str.lower de sort() 3. Por motivos técnicos, el método sort() pone los caracteres en mayúscula como Z antes de caracteres en minúscula como a; el argumento que proporcionamos evita esto comparando la forma en minúscula de las cadenas. Creamos un objeto PdfWriter para guardar las páginas de los PDF combinados 4. Por último, unos cuantos comentarios esbozan el resto del programa.

Paso 2: abrir cada PDF

Ahora, el programa debe leer cada archivo PDF de pdf_filenames. Añade esto al programa:

```
# combine_pdfs.py - Combina todos los PDF del directorio de trabajo actual
```

```
# en un único archivo PDF
```

```
import pypdf, os
```

```
--fragmento omitido--
```

```
# Pasa en bucle por todos los archivos PDF:
```

```
for pdf_filename in pdf_filenames:
```

```
    reader = pypdf.PdfReader(pdf_filename)
```

Copia todas las páginas después de la primera:

writer.append(pdf_filename, (1, len(reader.pages)))

PENDIENTE: Guardar el PDF resultante en un archivo.

Para cada nombre de archivo PDF, el bucle crea un objeto PdfReader y lo guarda en una variable llamada reader. Ahora el código que está dentro del bucle puede llamar a len(reader.pages) para averiguar cuántas páginas tiene el PDF. Utiliza esta información en la llamada al método append() para copiar las páginas a partir de la 1 (que es la segunda porque PyPDF usa 0 como índice de la primera página) hasta el final del PDF. Después, añade el contenido al mismo objeto PdfWriter de writer.

Paso 3: guardar los resultados

Cuando los bucles for terminen, la variable writer debería contener un objeto PdfWriter con las páginas de todos los objetos PDF combinados. El último paso es escribir este contenido en un archivo del disco duro. Añade este código al programa:

combine_pdfs.py - Combina todos los PDF del directorio de trabajo actual

en un único archivo PDF

import pypdf, os

--fragmento omitido--

Guarda el PDF resultante en un archivo:

with open('combined.pdf', 'wb') as file:

writer.write(file)

Pasar 'wb' a open() abre el archivo PDF de salida, combined.pdf, en modo escritura binario. Después, pasar el objeto File resultante al método write() crea el archivo PDF real. (Fíjate en los métodos write() que tienen el mismo nombre en los objetos File y PdfWriter). Al final del programa, un solo PDF contiene todas las

páginas (excepto la primera) de todos los PDF de la carpeta, ordenados alfabéticamente por nombre de archivo.

Ideas para programas similares

Ser capaz de crear un PDF a partir de páginas de otros PDF te permitirá escribir programas que hagan tareas como estas:

- Cortar páginas específicas de archivos PDF.
- Invertir o reordenar páginas en un PDF.
- Crear un PDF solo con las páginas de otros PDF que tengan texto específico, identificado por el método `extract_text()` de los objetos `Page`.

Documentos de Word

Python puede crear y modificar documentos de Microsoft Word, que tienen la extensión `.docx`, con el paquete `Python-Docx`, que puedes instalar siguiendo las instrucciones del apéndice A.

Advertencia *Asegúrate de instalar `Python-Docx`, no `Docx`, que pertenece a otro paquete que no vamos a tratar en este libro. Sin embargo, al importar el módulo desde el paquete `Python-Docx`, tendrás que ejecutar `import docx`, y no `import python-docx`.*

Si no tienes Word, puedes utilizar la aplicación gratuita LibreOffice Writer para Windows, macOS y Linux para abrir archivos `.docx`. Descárgala en <https://www.libreoffice.org>. Aunque Word pueda ejecutarse en macOS, en este capítulo nos centraremos en Word para Windows. Recuerda también que las aplicaciones basadas en navegador Office 365 y Google Docs son procesadores de texto populares y también pueden importar y exportar archivos `.docx`.

En comparación con los archivos de texto simple, los `.docx` tienen muchos elementos estructurales que `Python-Docx` representa con tres tipos de datos diferentes. En el nivel superior, un objeto `Document` representa el documento entero. El objeto `Document` contiene una lista de objetos `Paragraph` para los párrafos del documento. (Un párrafo nuevo comienza cada vez que el usuario

pulsa **Intro** al escribir en un documento de Word). Cada uno de estos objetos Paragraph contiene una lista de uno o varios Run. El párrafo de una sola oración de la figura 17.4 tiene cuatro elementos.

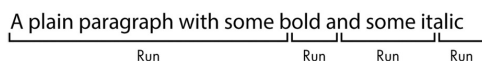


Figura 17.4. Los objetos Run identificados en un objeto Paragraph.

El texto de un documento de Word es más que una simple cadena. Tiene asociada información de fuente, tamaño, color y otros estilos. Un estilo de Word es una colección de tales atributos. Un objeto Run es un fragmento de texto corrido con el mismo estilo. Necesitarás un nuevo objeto Run cada vez que cambie el estilo del texto.

Leer documentos de Word

Vamos a experimentar con el módulo docx. Descarga el archivo demo.docx de la página de recursos en línea del libro y guárdalo en el directorio de trabajo actual. A continuación, introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > import docx
```

```
> > > doc = docx.Document('demo.docx')
```

```
> > > len(doc.paragraphs)
```

```
7
```

```
> > > doc.paragraphs[0].text
```

```
'Document Title'
```

```
> > > doc.paragraphs[1].text
```

```
'A plain paragraph with some bold text and some italic'
```

```
> > > len(doc.paragraphs[1].runs)
```

```
> > > doc.paragraphs[1].runs[0].text
```

```
'A plain paragraph with some '
```

```
> > > doc.paragraphs[1].runs[1].text
```

```
'bold'
```

```
> > > doc.paragraphs[1].runs[2].text
```

```
' and some '
```

```
> > > doc.paragraphs[1].runs[3].text
```

```
'italic'
```

Abrimos un archivo .docx en Python, llamamos a `docx.Document()` y le pasamos el nombre de archivo `demo.docx`. Esto devolverá un objeto `Document` con un atributo `paragraphs`, que es una lista de objetos `Paragraph`. Cuando llamamos a `len()` con este atributo, devuelve 7, lo cual nos dice que hay siete objetos `Paragraph` en este documento. Cada uno de estos objetos `Paragraph` tiene un atributo `text` que contiene una cadena del texto de ese párrafo (sin información de estilo). Aquí, el primer atributo `text` contiene 'DocumentTitle' y, el segundo, 'A plain paragraph with some bold text and some italic'.

Cada objeto `Paragraph` también tiene un atributo `run`, que es una lista de objetos `Run`. Los objetos `Run` tienen asimismo un atributo `text`, que contiene solo el texto de ese elemento en concreto. Echemos un vistazo a los atributos `text` del segundo objeto `Paragraph`. Llamar a `len()` en este objeto nos dice que hay cuatro objetos `Run`. El primero contiene 'A plain paragraph with some '. Después, el texto cambia a negrita, así que 'bold' empieza un nuevo objeto `Run`. El texto vuelve a redonda después, lo que produce un tercer objeto `Run`: ' text and some '. Por último, el cuarto y último objeto `Run` contiene 'italic' en cursiva.

Con Python-Docx, tus programas de Python podrán leer texto de un archivo .docx y usarlo como cualquier otro valor de cadena.

Si solo te importa el texto de un documento de Word y te da igual la información de estilo, puedes usar la función `get_text()`, que acepta el nombre de archivo de un .docx y devuelve un único valor de cadena de su texto. Abre una nueva pestaña en el editor de archivos, escribe el siguiente código y guárdalo como `readDocx.py`:

```
import docx

def get_text(filename):

    doc = docx.Document(filename)

    full_text = []

    for para in doc.paragraphs:

        full_text.append(para.text)

    return '\n'.join(full_text)
```

Esta función `get_text()` abre el documento de Word, pasa en bucle por todos los objetos `Paragraph` de la lista `paragraphs` y añade su texto a la lista `full_text`. Después del bucle, el código une las cadenas de `full_text` con saltos de línea. Podemos importar el programa `readDocx.py` como cualquier otro módulo. Si solo necesitamos el texto de un documento de Word, podemos introducir lo siguiente:

```
> > > import readDocx

> > > print(readDocx.get_text('demo.docx'))
```

Document Title

A plain paragraph with some bold text and some italic

Heading, level 1

Intense quote

first item in unordered list

first item in ordered list

También podemos ajustar `get_text()` para modificar la cadena antes de devolverla. Por ejemplo, para sangrar cada párrafo, sustituye la llamada a `append()` en `readDocx.py` por esto:

```
full_text.append(' ' + para.text)
```

Para añadir un doble espacio entre párrafos, cambia el código de la llamada a `join()` por esto:

```
return '\n\n'.join(full_text)
```

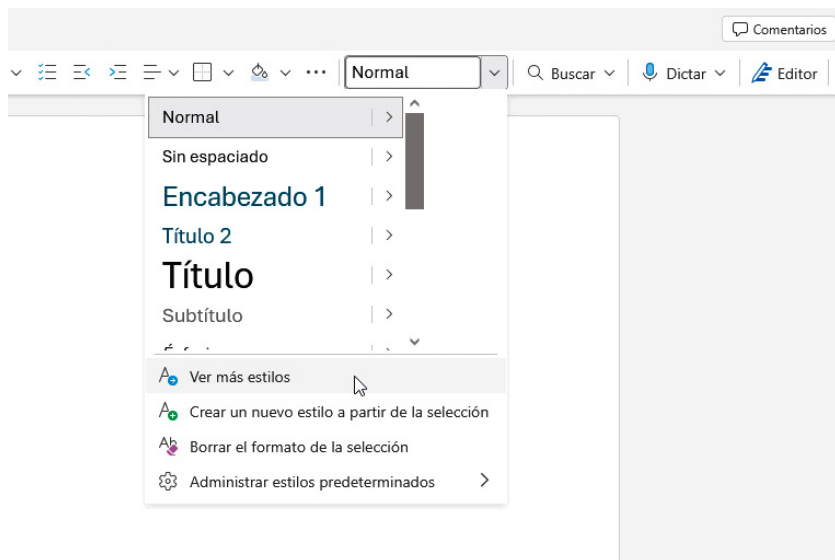
Como verás, solo requiere unas líneas de código escribir funciones que lean un archivo `.docx` y devuelvan una cadena de su contenido a tu gusto.

Dar estilo a objetos Paragraph y Run

Word y otros procesadores de texto utilizan estilos para dar consistencia visual a la presentación del texto y facilitar los cambios. Por ejemplo, puede que queramos que todos los párrafos del cuerpo de texto estén en Times New Roman 11, alineados a la izquierda y sin justificar. Podemos crear un estilo con esa configuración y asignárselo a todos los párrafos de cuerpo. Si luego queremos cambiar la presentación de todos los párrafos, podemos cambiar el estilo para actualizarlos automáticamente. Para ver los estilos en la aplicación Word de Office 365, haz clic en el menú Inicio y selecciona el desplegable Títulos y otros estilos, que seguramente mostrará Normal u otro nombre de estilo. Haz clic en Ver más estilos para abrir la ventana Más estilos, como se muestra en la figura 17.5. En la aplicación de escritorio Microsoft Word para Windows, puedes ver todos los estilos pulsando **Control-Alt-Mayús-S** para mostrar el panel Estilos. En LibreOffice Writer, puedes ver el panel Estilos haciendo clic en el menú Ver > Estilos.

Los documentos de Word contienen tres tipos de estilos: de párrafo, que se aplican a los objetos Paragraph, de carácter, que se aplican a los objetos Run y estilos asociados que se aplican a ambos tipos de objetos. Para dar estilo a objetos Paragraph y Run, estableceremos su atributo `style` en una cadena del nombre del estilo. Si `style` es

'New England' 5'
 'New England' 2'
 'New England' 2'
 'New England' 18'
 'New England' 2'
 'New England' 3'
 'New England' 2'
 'New England' 3'



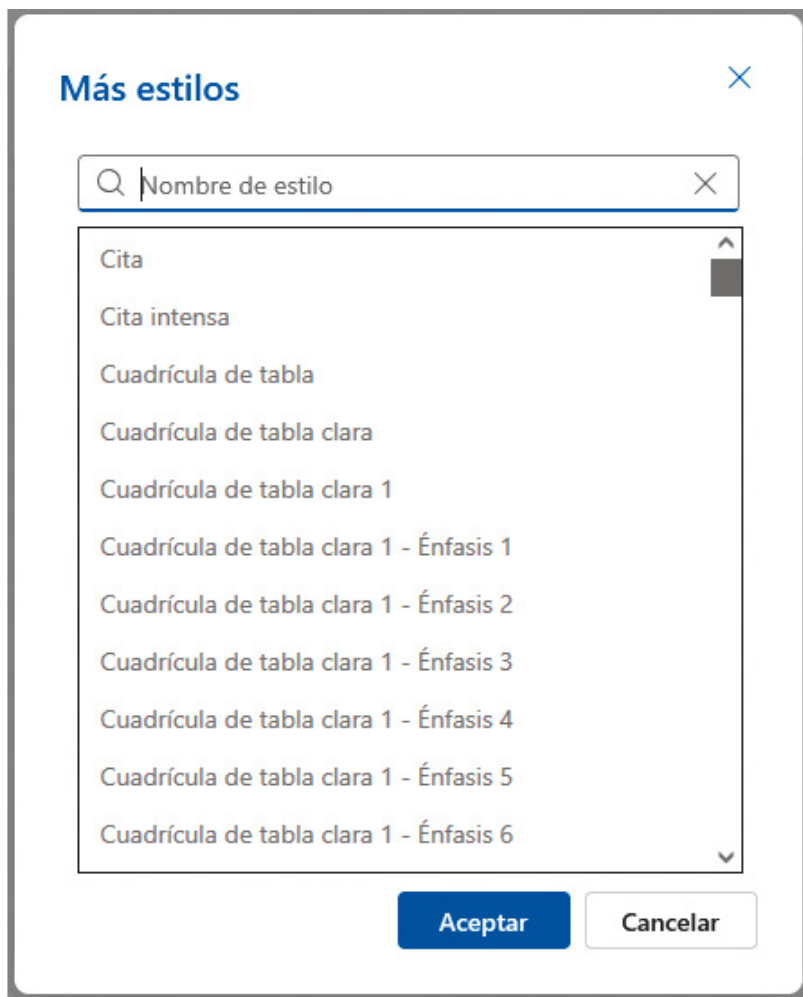


Figura 17.5. El panel de estilos y la ventana Más estilos.

Cuando se usa un estilo asociado para un objeto Run, hay que añadir 'Char' al final del nombre. Por ejemplo, para usar el estilo Cita para un objeto Paragraph, usaremos `paragraphObj.style = 'Quote'`, pero, para un objeto Run, será `runObj.style = 'Quote Char'`. Para crear estilos personalizados, utiliza la aplicación Word para definirlos y luego lee el atributo `style` de un objeto Paragraph o Run.

Podemos dar más estilo a elementos con atributos text. Cada atributo se puede establecer en uno de tres valores: True (que significa que el atributo está siempre habilitado, al margen de qué otros estilos se apliquen), False (que significa que el atributo está siempre deshabilitado) o None (que deja por defecto el estilo en el que está el elemento). La tabla 17.1 recoge los atributos text que se pueden establecer con objetos Run.

Tabla 17.1. Atributos text de objetos Run.

	Descripción
Bold	El texto se pone en negrita.
Italic	El texto se pone en cursiva.
Underline	El texto se subraya.
Strike	El texto aparece tachado.
DoubleStrike	El texto aparece con doble tachado.
All caps	El texto aparece en mayúsculas.
Small caps	El texto aparece en mayúsculas, con las minúsculas dos puntos más pequeñas.
Shadow	El texto aparece con una sombra.
Outline	El texto aparece como contorno sin relleno.
Right	El texto se escribe de derecha a izquierda.
Emboss	El texto aparece grabado en la página.
Relief	El texto aparece en relieve.

Por ejemplo, para cambiar los estilos de demo.docx, introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > import docx
```

```
> > > doc = docx.Document('demo.docx')
```

```
> > > doc.paragraphs[0].text
```

```
'Document Title'
```

```
> > > doc.paragraphs[0].style # El id exacto puede ser
diferente.
```

```
_ParagraphStyle('Title') id: 3095631007984
```

```
> > > doc.paragraphs[0].style = 'Normal'
```

```
> > > doc.paragraphs[1].text
```

‘A plain paragraph with some bold text and some italic’

```
> > > (doc.paragraphs[1].runs[0].text,  
doc.paragraphs[1].runs[1].text,
```

```
doc.paragraphs[1].runs[2].text,  
doc.paragraphs[1].runs[3].text)
```

(‘A plain paragraph with some ‘, ‘bold’, ‘ and some ‘, ‘italic’)

```
> > > doc.paragraphs[1].runs[0].style = ‘Quote Char’
```

```
> > > doc.paragraphs[1].runs[1].underline = True
```

```
> > > doc.paragraphs[1].runs[3].underline = True
```

```
> > > doc.save(‘restyled.docx’)
```

Usamos los atributos text y style para ver fácilmente los párrafos en el documento. Como verás, es fácil dividir un párrafo en elementos y acceder a ellos por separado. Obtenemos el primero, el segundo y el cuarto del segundo párrafo, les damos estilo y guardamos el resultado en un nuevo documento.

Ahora, las palabras “Document Title” al principio de restyled.docx deberían tener el estilo Normal en vez de Título, el objeto Run del texto “A plain paragraph with some” debería tener el estilo Cita y los dos objetos Run para las palabras “bold” e “italic” deberían tener sendos atributos underline establecidos en True. La figura 17.6 muestra cómo quedan los estilos de párrafos y elementos en restyled.docx.

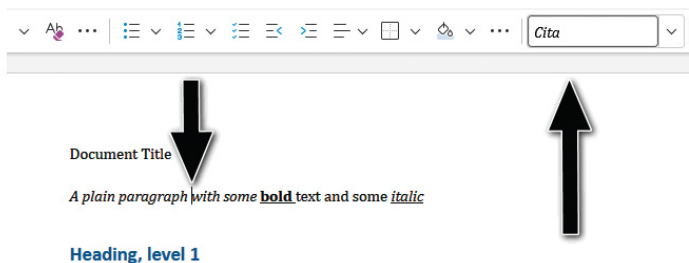


Figura 17.6. El archivo restyled.docx.

Encontrarás documentación completa sobre el uso de estilos de Python-Docx en <https://python-docx.readthedocs.io>.

Escribir documentos de Word

Para crear tu propio archivo .docx, llama a `docx.Document()` para que devuelva un nuevo objeto Word Document en blanco. Por ejemplo, introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > import docx

> > > doc = docx.Document()

> > > doc.add_paragraph('Hello, world!')

<docx.text.paragraph.Paragraph object at 0x0000000003B56F60>

> > > doc.save('helloworld.docx')
```

El método de documento `add_paragraph()` añade un nuevo párrafo de texto al documento y devuelve una referencia al objeto `Paragraph` que se ha añadido. Cuando añadimos texto, pasamos una cadena de nombre de archivo al método `save()` para guardar el objeto `Document` en un archivo.

Este código creará un archivo llamado `helloworld.docx` en el directorio de trabajo actual. Al abrirlo, debería quedar como la figura 17.7. Puedes cargar este archivo .docx en Office 365 o Google Docs o abrirlo en Word o LibreOffice.

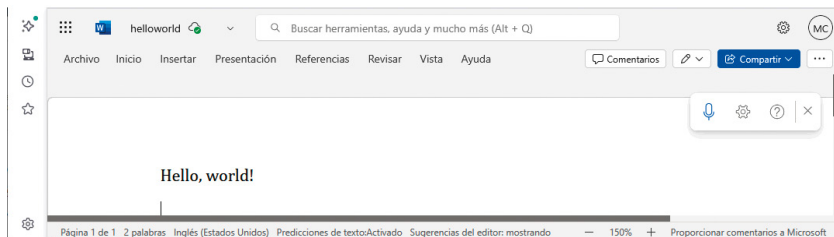


Figura 17.7. El documento de Word creado con

```
add_paragraph('Hello, world!').
```

Podemos añadir párrafos al documento llamando al método `add_paragraph()` otra vez con el texto del nuevo párrafo. Para añadir texto al final de un párrafo que ya existe, llama al método `add_run()` del párrafo y pásale una cadena. Introduce esto en el intérprete de comandos interactivo:

```
> > > import docx
```

```
> > > doc = docx.Document()
```

```
> > > doc.add_paragraph('Hello world!')
```

```
<docx.text.paragraph.Paragraph object at  
0x000000000366AD30 >
```

```
> > > para_obj_1 = doc.add_paragraph('This is a second  
paragraph.')
```

```
> > > para_obj_2 = doc.add_paragraph('This is a yet another  
paragraph.')
```

```
> > > para_obj_1.add_run(' This text is being added to the  
second paragraph.')
```

```
<docx.text.run.Run object at 0x0000000003A2C860 >
```

```
> > > doc.save('multipleParagraphs.docx')
```

El documento resultante debería ser como la figura 17.8. Observa que el texto “This text is being added to the second paragraph.” se ha añadido al objeto `Paragraph` de `para_obj_1`, que fue el segundo párrafo añadido al documento. Las funciones `add_paragraph()` y `add_run()` devuelven objetos `Paragraph` y `Run`, respectivamente, para evitarnos el problema de extraerlos en un paso aparte. Llama al método `save()` de nuevo para guardar los cambios adicionales que has realizado.

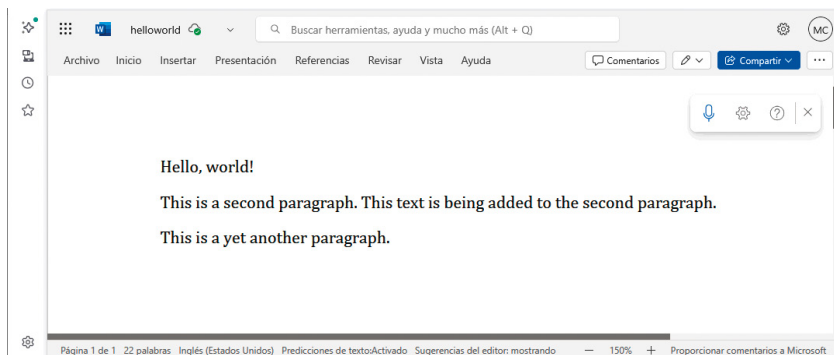


Figura 17.8. El documento con varios objetos Paragraph y Run añadidos.

Tanto `add_paragraph()` como `add_run()` aceptan un segundo argumento opcional que es una cadena del estilo del objeto Paragraph o Run. He aquí un ejemplo:

```
> > > doc.add_paragraph('Hello, world!', 'Title')
```

```
<docx.text.paragraph.Paragraph object at 0x00000213E6FA9190 >
```

Esta línea añade un párrafo con el texto “Hello, world!” en estilo Título.

Añadir encabezados

Llamar a `add_heading()` añade un párrafo con uno de los estilos de encabezado. Introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > import docx
```

```
> > > doc = docx.Document()
```

```
> > > doc.add_heading('Header 0', 0)
```

```
<docx.text.paragraph.Paragraph object at 0x00000000036CB3C8 >
```

```
> > > doc.add_heading('Header 1', 1)
```

```
<docx.text.paragraph.Paragraph object at 0x00000000036CB630 >
```

```
> > > doc.add_heading('Header 2', 2)
```

```
<docx.text.paragraph.Paragraph object at 0x00000000036CB828 >
```

```
> > > doc.add_heading('Header 3', 3)
```

```
<docx.text.paragraph.Paragraph object at 0x00000000036CB2E8 >
```

```
> > > doc.add_heading('Header 4', 4)
```

```
<docx.text.paragraph.Paragraph object at 0x00000000036CB3C8 >
```

```
> > > doc.save('headings.docx')
```

El archivo `headings.docx` resultante debería ser como la figura 17.9.

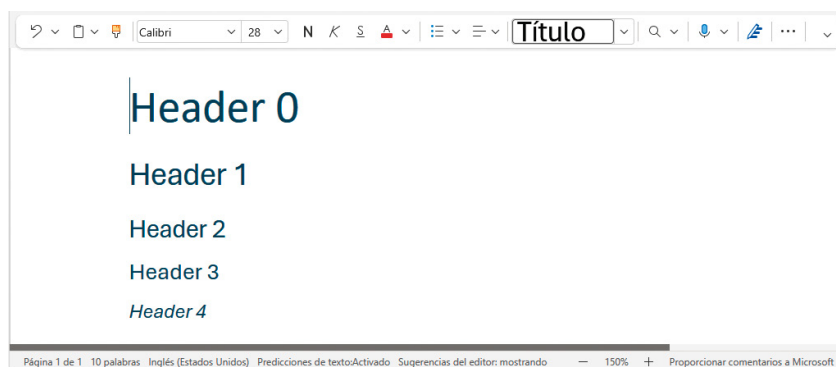


Figura 17.9. El documento `headings.docx` con encabezados de 0 a 4.

Los argumentos de `add_heading()` son una cadena que contiene el texto del encabezado y un entero del 0 al 4. El 0 pone el encabezado en estilo `Título`, que utilizamos para el principio del documento. Los enteros de 1 a 9 son para los distintos niveles de encabezado, siendo el 1 el más alto y el 9 el más bajo. La función `add_heading()` devuelve un objeto `Paragraph` para evitarnos extraerlo del objeto `Document` como un paso aparte.

Añadir saltos de línea y de página

Para añadir un salto de línea (en vez de empezar un párrafo nuevo),

podemos llamar al método `add_break()` con el objeto `Run` después del cual queremos introducir el salto. Si queremos añadir un salto de página, tendremos que pasar el valor `docx.enum.text.WD_BREAK.PAGE` como único argumento a `add_break()`, como en medio de este ejemplo:

```
> > > doc = docx.Document()

> > > doc.add_paragraph("This is on the first page!")

<docx.text.paragraph.Paragraph object at 0x0000000003785518 >

1 > > >
doc.paragraphs[0].runs[0].add_break(docx.enum.text.WD_BREAK.PAGE)

> > > doc.add_paragraph("This is on the second page!")

<docx.text.paragraph.Paragraph object at 0x00000000037855F8 >

> > > doc.save('twoPage.docx')
```

Este código crea un documento de Word de dos páginas con “This is on the first page!” en la primera página y “This is on the second page!” en la segunda. Incluso aunque haya mucho sitio en la primera página después del texto, hemos obligado al siguiente párrafo a empezar en otra página insertando un salto de página después del primer elemento del primer párrafo 1.

Añadir imágenes

Podemos usar el método `add_picture()` de los objetos `Document` para añadir una imagen al final del documento. Supongamos que tenemos un archivo `zophie.png` en el directorio de trabajo actual. Podemos añadir `zophie.png` al final del documento con una anchura de una pulgada y una altura de cuatro centímetros (Word puede usar los dos sistemas métricos) introduciendo lo siguiente:

```
> > > doc.add_picture('zophie.png',
width = docx.shared.Inches(1), height = docx.shared.Cm(4))

<docx.shape.InlineShape object at 0x00000000036C7D30 >
```

El primer argumento es una cadena con el nombre de archivo de la imagen. Los argumentos opcionales `width` y `height` establecerán la anchura y la altura de la imagen en el documento. Si no se incluyen, el alto y el ancho serán por defecto el tamaño normal de la imagen.

Es probable que prefieras especificar la altura y anchura de una imagen con unidades familiares, como pulgadas o centímetros, así que puedes usar las funciones `docx.shared.Inches()` y `docx.shared.Cm()` al especificar los argumentos `width` y `height`.

Resumen

La información textual no está solo en archivos de texto simple; de hecho, es bastante probable que las veas con mayor frecuencia con documentos en PDF o Word. Puedes usar el paquete PyPDF para leer y escribir archivos PDF, pero también hay librerías de Python que pueden hacerlo. Si prefieres usar alguna de las que no hemos comentado en este capítulo, te recomiendo buscar `pdfplumber`, `ReportLab`, `pdfcrow`, `PyMuPDF`, `pdfkit` y `borb` en el sitio web de PyPI.

Desafortunadamente, leer texto de documentos en PDF podría no producir una traducción perfecta a cadena porque el formato de archivo es complicado y algunos PDF pueden ser completamente ilegibles. El paquete `pdfminer.six` es un derivado de un paquete `pdfminer` ya abandonado que se centra en la extracción de texto de PDF. Este capítulo utiliza `pdfminer.six` como mecanismo de rescate si no se puede extraer texto de un archivo PDF en particular.

Los documentos de Word son más fiables y podemos leerlos con el módulo `docx` del paquete `python-docx`. Podemos manipular texto en documentos de Word mediante objetos `Paragraph` y `Run`. Estos objetos también pueden recibir estilos, aunque deben estar en el conjunto de estilos predeterminados o estar ya presentes en el documento. Podemos añadir párrafos nuevos, encabezados, saltos e imágenes al final de documentos.

Muchas de las limitaciones que implica el trabajo con documentos en PDF y Word vienen de que estos formatos están pensados para quedar bien ante lectores humanos, más que para ser analizados por

software. El próximo capítulo echa un vistazo a otros formatos habituales para guardar información: CSV, JSON y XML. Estos formatos se diseñaron para usarse con ordenadores y verás que Python puede trabajar con ellos con mucha más facilidad.

Preguntas prácticas

1. ¿En qué modo necesita abrirse el objeto File del objeto PdfWriter para guardar el archivo PDF?
2. ¿Cómo adquirimos un objeto Page para la página 5 de un objeto PdfReader o PdfWriter?
3. Si el PDF de un objeto PdfReader está encriptado con la contraseña swordfish, ¿qué hay que hacer antes de poder obtener objetos Page de él?
4. Si el método rotate() rota páginas en el sentido de las agujas del reloj, ¿cómo las giramos en sentido contrario?
5. ¿Qué método devuelve un objeto Document para un archivo llamado demo.docx?
6. ¿Cuál es la diferencia entre un objeto Paragraph y un objeto Run?
7. ¿Cómo podemos obtener una lista de objetos Paragraph para un objeto Document almacenado en una variable llamada doc?
8. ¿Qué tipo de objeto tiene variables bold, underline, italic, strike y outline?
9. ¿Cuál es la diferencia entre establecer la variable bold en True, False o None?
10. ¿Cómo creamos un objeto Document para un nuevo documento de Word?
11. ¿Cómo añadimos un párrafo con el texto 'Hello, there!' a un objeto Document almacenado en una variable llamada doc?
12. ¿Qué enteros representan los niveles de encabezado disponibles en los documentos de Word?

Programas de práctica

Para practicar, escribe programas que hagan las siguientes tareas.

Paranoia PDF

Con la función `os.walk()` del capítulo 11, escribe un *script* que pase por todos los PDF de una carpeta (y sus subcarpetas) y encripte los PDF con una contraseña proporcionada en la línea de comandos. Guarda cada PDF encriptado con un sufijo `_encrypted.pdf` añadido al nombre original. Antes de borrar el archivo original, haz que el programa intente leer y desencriptar el archivo nuevo para asegurarte de que está bien encriptado.

Después, escribe un programa que busque todos los PDF encriptados de una carpeta (y sus subcarpetas) y cree una versión desencriptada con una contraseña dada. Si la contraseña es incorrecta, el programa debería mostrar un mensaje al usuario y seguir con el siguiente PDF.

Invitaciones personalizadas

Supongamos que tienes un archivo de texto con nombres de invitados. Este archivo `guests.txt` tiene un nombre por línea, así:

Prof. Plum

Miss Scarlet

Col. Mustard

Al Sweigart

RoboCop

Escribe un programa que genere un documento de Word con invitaciones personalizadas como la que muestra la figura 17.10.

Como Python-Docx solo puede usar estilos que ya existan en un documento de Word, primero tendrás que añadir estos estilos a un Word en blanco y después abrir ese archivo con Python-Docx.

Debería haber una invitación por página en el Word resultante, así que llama a `add_break()` para añadir un salto de página después del último párrafo de cada invitación. De ese modo, solo necesitarás abrir un documento para imprimir todas las invitaciones a la vez.

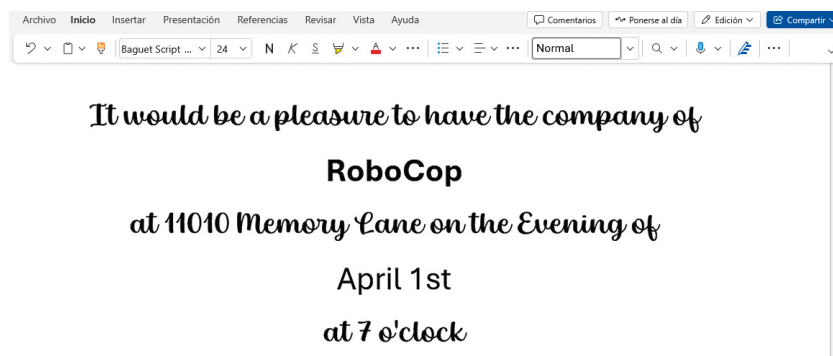


Figura 17.10. El documento de Word generado por el *script* de invitaciones personalizadas.

Puedes descargar un archivo `guests.txt` de muestra en los recursos en línea del libro.

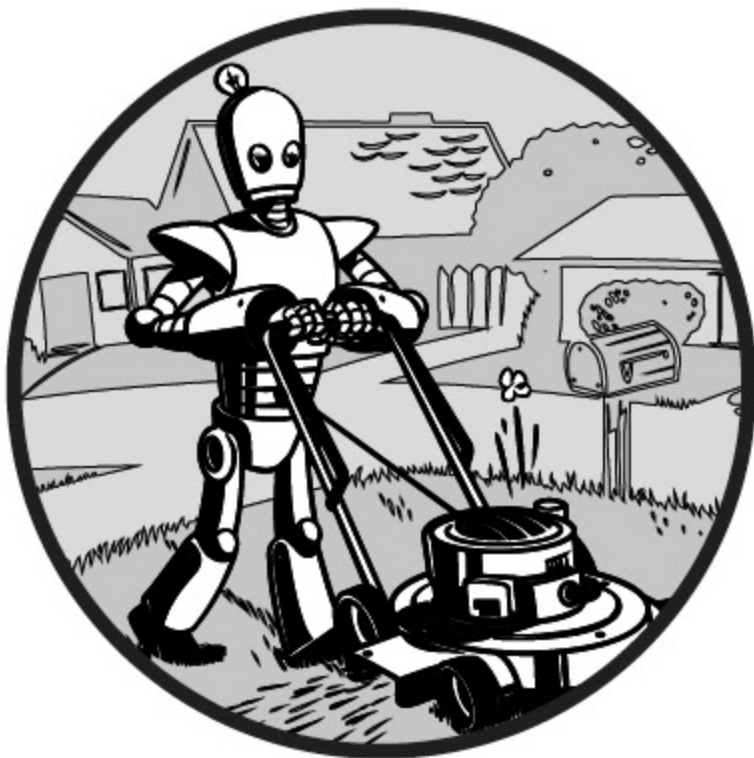
Revienta contraseñas de PDF

Supongamos que tienes un PDF encriptado cuya contraseña has olvidado, pero recuerdas que era una sola palabra en inglés. Intentar recordar la contraseña olvidada es una pesadez, así que, en vez de eso, vas a escribir un programa que descrypte el PDF probando con todas las palabras en inglés posibles hasta dar con la que funciona. Esto se denomina “ataque de fuerza bruta”. Descarga el archivo `dictionary.txt` de los recursos en línea del libro. Este diccionario contiene más de 44.000 palabras en inglés, una por línea.

Con las habilidades de lectura de archivos que aprendiste en el capítulo 10, crea una lista de cadenas de palabra leyendo este archivo. Después, pasa en bucle por cada palabra de la lista, pasándosela al método `decrypt()`. Deberías probar tanto con mayúsculas como con minúsculas para cada palabra. (En mi portátil, pasar por las 88.000 formas del diccionario tarda un par de

minutos. Esta es la razón por la que nunca deberías usar una simple palabra como contraseña).

Archivos CSV, JSON y XML



CSV, JSON y XML son formatos de serialización de datos que sirven para almacenar información como archivos de texto simple. La serialización convierte los datos en una cadena para guardar el trabajo de nuestros programas en un archivo de texto, transferirlo por una conexión a Internet o incluso copiarlo y pegarlo en un correo electrónico. Python incluye los módulos `csv`, `json` y `xml` para ayudarnos a trabajar con estos formatos de archivo.

Aunque los archivos en estos formatos son, en esencia, archivos de texto que se podrían leer y escribir con la función `open()` de Python u otra de las funciones explicadas en el capítulo 10, es más fácil

utilizar módulos de Python para manejarlos, igual que usamos Beautiful Soup en el capítulo 13 para manejar archivos en formato HTML. Cada formato tiene su propio caso de uso:

•**Valores separados por comas (CSV):** Es un formato de hoja de cálculo simplificado que funciona mejor para almacenar un número variable de filas de datos que comparten las mismas columnas.

•**Notación de objetos JavaScript (JSON):** Usa la misma sintaxis de objetos, matrices y tipos de datos que el lenguaje de programación JavaScript, aunque no requiere saber programar en ese lenguaje. Se creó como alternativa sencilla a XML.

•**Lenguaje de marcado extensible (XML):** Es un formato de serialización de datos más antiguo y establecido que se emplea mucho en software empresarial, pero es demasiado complicado trabajar con él si no necesitas sus funciones avanzadas.

Este capítulo aborda lo básico de la sintaxis de estos formatos y el código Python para trabajar con ellos.

El formato CSV

Cada línea de un archivo CSV (que utiliza la extensión .csv) representa una fila en una hoja de cálculo y una coma separa las celdas en la fila. Por ejemplo, la hoja de cálculo example3.xlsx incluida en los recursos en línea del libro quedaría así en un archivo CSV:

4/5/2035 13:34,Apples,73

4/5/2035 3:41,Cherries,85

4/6/2035 12:46,Pears,14

4/8/2035 8:59,Oranges,52

4/10/2035 2:07,Apples,152

4/10/2035 18:10,Bananas,23

4/10/2035 2:40,Strawberries,98

Utilizaré este archivo en los ejemplos de intérprete de comandos interactivo de este capítulo para CSV. Descárgalo o escribe el texto en un editor de texto y guárdalo como `example3.csv`.

Puedes pensar en los archivos CSV como una lista de listas de valores. El código Python podría representar el contenido de `example3.csv` como el valor `[['4/5/2035 13:34', 'Apples', '73'], ['4/5/2035 3:41', 'Cherries', '85'], ... ['4/10/2035 2:40', 'Strawberries', '98']]`. Los archivos CSV son sencillos porque carecen de muchas de las características de las hojas de datos de Excel. Por ejemplo:

- No tienen múltiples tipos de datos; todos los valores son una cadena.
- No tienen configuraciones de tamaño ni color de fuente.
- No tienen varias hojas.
- No se puede especificar la anchura ni la altura de las celdas.
- No se puede fusionar celdas.
- No se puede incrustar gráficos ni imágenes.

La ventaja de los archivos CSV es su sencillez. Son compatibles con muchas aplicaciones y lenguajes de programación, pueden abrirse en editores de texto (incluido Mu) y son una forma fácil de representar datos de hoja de cálculo.

Como los CSV son solo archivos de texto, puedes caer en la tentación de leerlos como una cadena y procesar esa cadena con las técnicas aprendidas en el capítulo 8. Por ejemplo, como cada fila de un archivo CSV está separada por una coma, podrías intentar llamar a `split(',')` con cada línea de texto para obtener los valores separados por comas como una lista de cadenas. Pero no todas las comas de un CSV representan el límite entre dos celdas. Los archivos CSV tienen un conjunto de caracteres de escape que permiten incluir comas y otros caracteres como parte de los valores. El método `split()` no gestiona estos caracteres de escape. Por este tipo de problemas potenciales, el módulo `csv` ofrece una manera más fiable

de leer y escribir archivos CSV.

Leer archivos CSV

Para leer un archivo CSV, hay que crear un objeto `csv.reader`, que permite iterar por las líneas del archivo. El módulo `csv` está incluido en Python, así que puedes importarlo sin tener que instalarlo antes. Pon `example3.csv` en el directorio de trabajo actual e introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > import csv

> > > example_file = open('example3.csv')

> > > example_reader = csv.reader(example_file)

> > > example_data = list(example_reader)

> > > example_data

[['4/5/2035 13:34', 'Apples', '73'], ['4/5/2035 3:41', 'Cherries',
'85'],

['4/6/2035 12:46', 'Pears', '14'], ['4/8/2035 8:59', 'Oranges', '52'],

['4/10/2035 2:07', 'Apples', '152'], ['4/10/2035 18:10', 'Bananas',
'23'],

['4/10/2035 2:40', 'Strawberries', '98']]

> > > example_file.close()
```

Para leer un archivo CSV con el módulo `csv`, ábrelo usando la función `open()`, como abrirías cualquier otro archivo de texto, pero, en lugar de llamar al método `read()` o `readlines()` con el objeto `File` que devuelve `open()`, pásaselo a la función `csv.reader()`. Esta función debería devolver un objeto `reader`. Observa que no se puede pasar una cadena de nombre de archivo directamente a la función `csv.reader()`.

La forma más fácil de acceder a los valores del objeto `reader` es convertirlo en una lista simple de Python pasándoselo a `list()`. Así,

este objeto reader devuelve una lista de listas que podemos almacenar en una variable, como `example_data`. Introducir `example_data` en el intérprete de comandos interactivo muestra la lista de listas.

Ahora que tenemos el archivo CSV como una lista de listas, podemos acceder al valor de una fila y columna en particular con la expresión `example_data[row][col]`, donde `row` es el índice de una de las listas de `example_data` y `col` el del elemento que queremos de esa lista. Escribe esto en el intérprete de comandos interactivo:

```
> > > example_data[0] [0] # Primera fila, primera columna
```

```
'4/5/2035 13:34'
```

```
> > > example_data[0] [1] # Primera fila, segunda columna
```

```
'Apples'
```

```
> > > example_data[0] [2] # Primera fila, tercera columna
```

```
'73'
```

```
> > > example_data[1] [1] # Segunda fila, segunda columna
```

```
'Cherries'
```

```
> > > example_data[6] [1] # Séptima fila, segunda columna
```

```
'Strawberries'
```

Como se aprecia en la salida, `example_data[0][0]` entra en la primera lista y nos da la primera cadena, `example_data[0][2]` entra en la primera lista y nos da la tercera cadena, etc.

Acceder a los datos de un bucle for

Para archivos CSV grandes, puede ser conveniente usar el objeto reader en un bucle for. Esta estrategia nos evita tener que cargar todo el archivo en memoria a la vez. Por ejemplo, introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > import csv

> > > example_file = open('example3.csv')

> > > example_reader = csv.reader(example_file)

1 > > > for row in example_reader:

. 2 print('Row ' + str(example_reader.line_num) + ' ' +
str(row))
```

Row #1 ['4/5/2035 13:34', 'Apples', '73']

Row #2 ['4/5/2035 3:41', 'Cherries', '85']

Row #3 ['4/6/2035 12:46', 'Pears', '14']

Row #4 ['4/8/2035 8:59', 'Oranges', '52']

Row #5 ['4/10/2035 2:07', 'Apples', '152']

Row #6 ['4/10/2035 18:10', 'Bananas', '23']

Row #7 ['4/10/2035 2:40', 'Strawberries', '98']

Después de importar el módulo csv y crear un objeto reader a partir del archivo CSV, podemos pasar en bucle por las filas del objeto reader 1. Cada fila es una lista de valores almacenados en la variable row y cada valor de la lista representa una celda.

La llamada a la función print() 2 imprime el número de la fila actual y su contenido. Para obtener el número de fila, utilizamos el atributo line_num del objeto reader, que guarda un entero. Si tu archivo CSV contiene encabezados de columna en la primera fila, podrías usar line_num para comprobar si estás en la fila 1 y ejecutar una instrucción continue para omitir los encabezados. A diferencia de los índices de lista de Python, los números de línea de line_num empiezan en 1, no en 0.

Solo se puede pasar en bucle por el objeto reader una vez. Para

releer el archivo CSV, hay que volver a llamar a `open()` y `csv.reader()` para crear otro objeto reader.

Escribir archivos CSV

Un objeto `csv.writer` permite escribir datos en un archivo CSV. Para crear un objeto writer, utiliza la función `csv.writer()`. Introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > import csv

1 > > > output_file = open('output.csv', 'w', newline='')

2 > > > output_writer = csv.writer(output_file)

> > > output_writer.writerow(['spam', 'eggs', 'bacon', 'ham'])

21

> > > output_writer.writerow(['Hello, world!', 'eggs', 'bacon',
'ham'])

32

> > > output_writer.writerow([1, 2, 3.141592, 4])

16

> > > output_file.close()
```

Llama a `open()` y pásale `'w'` para abrir el archivo en modo escritura 1. Este código debería crear un objeto que después pasamos a `csv.writer()` 2 para generar un objeto writer. En Windows, también habrá que pasar una cadena en blanco para el argumento `newline` de la función `open()`. Por razones técnicas que quedan fuera del ámbito de este libro, si olvidas establecer el argumento `newline`, las filas de `output.csv` quedarán con espacios entre medio, como se muestra en la figura 18.1.

	A	B	C	D	E	F
1	spam	eggs	bacon	ham		
2						
3	Hello, wor	eggs	bacon	ham		
4						
5	1	2	3.141592	4		
6						

Figura 18.1. Un archivo CSV con dobles espacios.

El método `writerow()` del objeto `writer` toma un argumento de lista. Cada valor de la lista aparecerá en su propia celda en el archivo CSV de salida. El valor de retorno del método es el número de caracteres escritos en el archivo para esa fila (incluidos los de salto de línea). En nuestro ejemplo, este código produce un archivo `output.csv` similar a esto:

spam,eggs,bacon,ham

“Hello, world!”,eggs,bacon,ham

1,2,3.141592,4

Observa cómo el objeto `writer` utiliza automáticamente las comillas dobles como carácter de escape para mantener la coma en el valor ‘Hello, world!’ del archivo CSV. El módulo `csv` nos evita tener que manejar nosotros estos casos especiales.

Usar tabuladores en vez de comas

Los archivos de valores separados por tabuladores (TSV) son similares a los CSV, pero, para sorpresa de nadie, usan tabuladores en lugar de comas. Estos archivos tienen la extensión `.tsv`. Supongamos que queremos separar celdas con un carácter de tabulación en lugar de una coma y que haya doble espacio entre las filas. Podríamos introducir esto en el intérprete de comandos interactivo:

```
> > import csv
```

```
> > output_file = open('output.tsv', 'w', newline='')
```

```

> > > output_writer = csv.writer(output_file, delimiter='\\t',
lineterminator='\\n\\n') 1

> > > output_writer.writerow(['spam', 'eggs', 'bacon', 'ham'])

21

> > > output_writer.writerow(['Hello, world!', 'eggs', 'bacon',
'ham'])

30

> > > output_writer.writerow([1, 2, 3.141592, 4])

16

> > > output_file.close()

```

Este código cambia los caracteres para delimitar y terminar la línea en el archivo. El delimitador es el carácter que aparece entre las celdas de una fila. Por defecto, en un CSV es una coma. El carácter para terminar la línea es el que aparece al final de una fila; por defecto, es un salto de línea. Se puede cambiar los caracteres por valores diferentes utilizando los argumentos `delimiter` y `lineterminator` con `csv.writer()`.

Pasar `delimiter='\\t'` and `lineterminator='\\n\\n'` 1 cambia el delimitador a un tabulador y el final de línea a dos saltos de línea. A continuación, el código llama a `writerow()` tres veces para crear tres filas, produciendo un archivo llamado `output.tsv` con el siguiente contenido:

```

ham
eggs
Hello, world!
3.141592

```

Ahora las celdas de la hoja de cálculo están separadas por tabuladores.

Manejar filas de encabezado

Para los archivos CSV que contienen filas de encabezado, suele ser

más cómodo trabajar con los objetos DictReader y DictWriter en lugar de reader y writer. Mientras que reader y writer leen y escriben en filas de archivos CSV empleando listas, DictReader y DictWriter realizan las mismas funciones con diccionarios, tratando los valores de la primera fila como las claves. Descarga `exampleWithHeader3.csv` de los recursos en línea del libro para el siguiente ejemplo. Este archivo es el mismo que `example3.csv`, pero incluye `Timestamp`, `Fruit` y `Quantity` como encabezados de columnas en la primera fila. Para leer el archivo, introduce esto en el intérprete de comandos interactivo:

```
> > > import csv
```

```
> > > example_file = open("exampleWithHeader3.csv")
```

```
> > > example_dict_reader = csv.DictReader(example_file)
```

```
1 > > > example_dict_data = list(example_dict_reader)
```

```
> > > example_dict_data
```

```
[{'Timestamp': '4/5/2035 3:41', 'Fruit': 'Cherries', 'Quantity': '85'},  
{'Timestamp': '4/6/2035 12:46', 'Fruit': 'Pears', 'Quantity': '14'},  
{'Timestamp': '4/8/2035 8:59', 'Fruit': 'Oranges', 'Quantity': '52'},  
{'Timestamp': '4/10/2035 2:07', 'Fruit': 'Apples', 'Quantity': '152'},  
{'Timestamp': '4/10/2035 18:10', 'Fruit': 'Bananas', 'Quantity':  
'23'},  
{'Timestamp': '4/10/2035 2:40', 'Fruit': 'Strawberries', 'Quantity':  
'98'}]
```

```
> > > example_file = open("exampleWithHeader3.csv")
```

```
> > > example_dict_reader = csv.DictReader(example_file)
```

```
2 > > > for row in example_dict_reader:
```

```
... print(row["Timestamp"], row["Fruit"], row["Quantity"])
```


.

4/5/2035	13:34	Apples	73
4/5/2035	3:41	Cherries	85
4/6/2035	12:46	Pears	14
4/8/2035	8:59	Oranges	52
4/10/2035	2:07	Apples	152
4/10/2035	18:10	Bananas	23
4/10/2035	2:40	Strawberries	98

Al pasar el objeto DictReader a list() 1, podemos obtener los datos del CSV como una lista de diccionarios. Cada fila corresponde a un diccionario de la lista. Como alternativa, podemos usar el objeto DictReader dentro de un bucle for 2. El objeto DictReader establece row en un objeto de diccionario con claves derivadas de los encabezados de la primera fila. Utilizar un objeto DictReader implica no necesitar código adicional para omitir la información del encabezado, ya que el objeto se encarga de eso.

Si intentásemos usar un objeto DictReader con example3.csv, que no tiene encabezados de columna en la primera fila, el objeto usaría '4/5/2035 13:34', 'Apples' y '73' como claves del diccionario. Para evitar esto, podemos dar a la función DictReader() un segundo argumento con nombres de encabezado inventados:

```
> > > import csv

> > > example_file = open('example3.csv')

> > > example_dict_reader = csv.DictReader(example_file,
['time', 'name', 'amount'])

> > > for row in example_dict_reader:

... print(row['time'], row['name'], row['amount'])
```

.

4/5/2035 13:34 Apples 73

4/5/2035 3:41 Cherries 85

4/6/2035 12:46 Pears 14

4/8/2035 8:59 Oranges 52

4/10/2035 2:07 Apples 152

4/10/2035 18:10 Bananas 23

4/10/2035 2:40 Strawberries 98

Como la primera fila de example3.csv no contiene encabezados de columna, hemos creado unos propios: 'time', 'name' y 'amount'. Los objetos DictWriter usan diccionarios para crear archivos CSV:

```
> > > import csv
```

```
> > > output_file = open('output.csv', 'w', newline='')
```

```
> > > output_dict_writer = csv.DictWriter(output_file,  
[ 'Name', 'Pet', 'Phone' ] )
```

```
> > > output_dict_writer.writeheader()
```

16

```
> > > output_dict_writer.writerow({'Name': 'Alice', 'Pet': 'cat',  
'Phone': '555-1234'})
```

20

```
> > > output_dict_writer.writerow({'Name': 'Bob', 'Phone':  
'555-9999'})
```

15

```
> > > output_dict_writer.writerow({'Phone': '555-5555',  
'Name': 'Carol', 'Pet': 'dog'})
```

> > > **output_file.close()**

Si quieres que tu archivo contenga una fila de encabezado, escríbela llamando a `writeheader()`. Si no, no llames a `writeheader()` para omitir la fila de encabezados del archivo. Después, puedes escribir cada fila del CSV con una llamada al método `writerow()`, pasando un diccionario que utilice los encabezados como claves y contenga los datos que escribir en el archivo. El archivo `output.csv` que crea este código debería ser:

Name,Pet,Phone

Alice,cat,555-1234

Bob,,555-9999

Carol,dog,555-5555

Las comas dobles indican que Bob tiene un valor en blanco para una mascota. Observa que el orden de los pares clave-valor del diccionario pasado a `writerow()` no importa; se escriben en el orden de las claves dadas a `DictWriter()`. Por ejemplo, aunque pasásemos el par clave-valor `Phone` antes de `Name` y `Pet` en la cuarta fila, el número de teléfono seguiría apareciendo el último en la salida.

Fíjate también en que cualquier clave que falte, como `'Pet'` en `{'Name': 'Bob', 'Phone': '555-9999'}`, se convertirá en una celda vacía en el archivo CSV.

Proyecto 13: Quitar el encabezado de archivos CSV

Imagina que te han encomendado la aburrida tarea de quitar la primera línea de cientos de archivos CSV. Puede que tengas que usarlos en un proceso automatizado que requiera solo los datos, sin los encabezados de las columnas. Podrías abrir cada archivo en Excel, borrar la primera fila y guardar el archivo, pero tardarías horas. Vamos a escribir un programa que lo haga por ti.

El programa necesitará abrir todos los archivos con la extensión `.csv`

en el directorio de trabajo actual, leer el contenido del archivo CSV y volver a escribir el contenido sin la primera fila en un archivo con el mismo nombre. Así se sustituirá el contenido antiguo del archivo CSV por este nuevo sin encabezados.

Advertencia *Como siempre, cuando escribas un programa que modifique archivos, asegúrate de hacer antes una copia de seguridad de los archivos por si el programa no funciona como esperabas. Evita borrar por accidente los archivos originales.*

En el nivel alto, el programa debe hacer lo siguiente:

- Buscar todos los archivos CSV del directorio de trabajo actual.
- Leer el contenido completo de cada archivo.
- Escribir el contenido, omitiendo la primera línea, en un nuevo archivo CSV.

A nivel de código, esto significa que el programa necesitará hacer lo siguiente:

- Pasar en bucle por una lista de archivos de `os.listdir()`, omitiendo los archivos que no sean CSV.
- Crear un objeto reader y leer el contenido del archivo CSV, con el atributo `line_num` para saber qué línea omitir.
- Crear un objeto writer y escribir los datos leídos en el nuevo archivo.

Para este proyecto, abre una nueva ventana del editor de archivos y guárdalo como `removeCsvHeader.py`.

Paso 1: pasar en bucle por todos los archivos

Lo primero que debe hacer el programa es pasar en bucle por una lista de todos los nombres de archivo CSV del directorio de trabajo actual. Empieza a escribir `removeCsvHeader.py` así:

```
# Elimina la línea de encabezados de archivos csv
```

```

import csv, os

os.makedirs('headerRemoved', exist_ok=True)

# Pasa en bucle por todos los archivos del directorio de trabajo
actual.

for csv_filename in os.listdir('.'):

    if not csv_filename.endswith('.csv'):

        1 continue # Omite los que no son CSV.

    print('Removing header from ' + csv_filename + '...')

    # PENDIENTE: Leer el archivo CSV (omitiendo la primera fila).

    # PENDIENTE: Escribir el archivo CSV.

```

La llamada a `os.makedirs()` crea una carpeta `headerRemoved` en la que guardar los archivos CSV sin los encabezados. Un bucle `for` en `os.listdir('.')` nos ayuda con eso, pero pasará por todos los archivos del directorio de trabajo actual, así que tendremos que añadir un poco de código al principio del bucle para saltarse los nombres de archivo que no terminen en `.csv`. La declaración `continue 1` hace que el bucle `for` avance hasta el siguiente nombre de archivo cuando llega a uno que no es CSV.

Para ver la salida mientras se ejecuta el programa, imprime un mensaje que indique en qué archivo CSV está trabajando. Después, añade comentarios PENDIENTE para describir lo que debería hacer el resto del programa.

Paso 2: leer el archivo

El archivo no elimina la primera fila del CSV, sino que crea una copia del archivo sin la primera línea. De ese modo, podemos usar el archivo original si un fallo modifica incorrectamente el archivo nuevo. El programa necesitará una forma de rastrear si está en ese momento pasando en bucle por la primera fila. Añade lo siguiente a `removeCsvHeader.py`.

```

# Elimina la línea de encabezados de archivos csv

import csv, os

--fragmento omitido--

# Lee el archivo CSV (omitiendo la primera fila).

csv_rows = []

csv_file_obj = open(csv_filename)

reader_obj = csv.reader(csv_file_obj)

for row in reader_obj:

    if reader_obj.line_num == 1:

        continue # Se salta la primera fila.

    csv_rows.append(row)

csv_file_obj.close()

# PENDIENTE: Escribir el archivo CSV.

```

El atributo `line_num` del objeto `reader` sirve para determinar qué fila del archivo CSV se está leyendo actualmente. Otro bucle `for` pasará por las filas devueltas del objeto `reader` y todas las filas, salvo la primera, se añadirán a `csv_rows`.

A medida que el bucle `for` itera por cada fila, el código comprueba si `reader_obj.line_num` está establecido en 1. Si lo está, ejecuta `continue` para pasar a la siguiente fila sin añadir la primera a `csv_rows`. Para cada línea siguiente, la condición será siempre `False`, por lo que el código añadirá la fila a `csv_rows`.

Paso 3: escribir el nuevo archivo CSV

Ahora que `csv_rows` contiene todas las filas menos la primera, necesitamos escribir la lista en un archivo CSV en la carpeta `headerRemoved`. Añade lo siguiente a `removeCsvHeader.py`:

```

# Elimina la línea de encabezados de archivos csv

import csv, os

--fragmento omitido--

# Pasa en bucle por todos los archivos del directorio de trabajo
actual.

1 for csv_filename in os.listdir('.'):

if not csv_filename.endswith('.csv'):

continue # Omite los que no son CSV.

--fragmento omitido--

# Escribe el archivo CSV.

csv_file_obj = open(os.path.join('headerRemoved',
csv_filename), 'w',

newline='')

csv_writer = csv.writer(csv_file_obj)

for row in csv_rows:

csv_writer.writerow(row)

csv_file_obj.close()

```

El objeto writer escribirá la lista en un archivo CSV en headerRemoved con csv_filename (que también hemos usado en el objeto reader). Una vez creado el objeto writer, pasamos en bucle por las sublistas almacenadas en csv_rows y las vamos escribiendo en el archivo.

El bucle for exterior 1 pasará al siguiente nombre de archivo devuelto por os.listdir('.'). Cuando termine ese bucle, se completará el programa.

Para probar el programa, descarga removeCsvHeader.zip de los

recursos en línea del libro y descomprímelo en una carpeta. A continuación, ejecuta `removeCsvHeader.py` en esa carpeta. La salida será así:

Removing header from NAICS_data_1048.csv...

Removing header from NAICS_data_1218.csv...

--*fragmento omitido*--

Removing header from NAICS_data_9834.csv...

Removing header from NAICS_data_9986.csv...

Este programa debería imprimir un nombre de archivo cada vez que quita la primera línea de un archivo CSV.

Ideas para programas similares

Los programas que trabajan con archivos CSV son similares a los que trabajan con archivos de Excel, ya que ambos son hojas de cálculo. Podrías escribir programas que hagan lo siguiente:

- Comparar datos entre distintas filas de un archivo CSV o entre varios archivos CSV.
- Copiar datos específicos de un archivo CSV a un archivo de Excel o viceversa.
- Comprobar si hay datos no válidos o errores de formato en archivos CSV y avisar al usuario de tales errores.
- Leer datos de un archivo CSV como entrada para otros programas de Python.

Formatos de texto simple versátiles

Mientras que los CSV son útiles para almacenar filas de datos que tienen exactamente las mismas columnas, los formatos JSON y XML pueden almacenar diversas estructuras de datos. (Este libro se salta los formatos YAML y TOML, menos populares pero igualmente

útiles). Estos formatos no son específicos de Python; muchos lenguajes de programación tienen funciones para leer y escribir datos en ellos.

Cada uno de estos formatos organiza los datos con el equivalente de los diccionarios y listas de Python. En otros lenguajes de programación, verás diccionarios denominados asignaciones, mapas de *hash*, tablas de *hash* o matrices asociativas (porque asignan o asocian un dato, la clave, con otro, el valor). Del mismo modo, verás que otros programas llaman matrices a las listas de Python, pero los conceptos son los mismos: organizan datos en pares clave-valor y listas.

Se puede anidar diccionarios y listas dentro de otros diccionarios y listas para formar estructuras de datos elaboradas. Pero, si queremos guardar estas estructuras de datos en un archivo de texto, habrá que elegir un formato de serialización de datos como JSON o XML. Los módulos de Python de este capítulo pueden analizar (es decir, leer y entender) texto escrito en estos formatos para crear estructuras de datos de Python a partir de esos textos.

Estos formatos de texto simple legibles por humanos no hacen un uso precisamente eficiente del espacio en disco o la memoria, pero tienen la ventaja de ser fáciles de ver y editar en un editor de texto y no distinguen lenguajes, ya que un programa escrito en cualquier lenguaje de programación puede leer o escribir este tipo de archivos. Por contra, el módulo *shelve*, del que hablamos en el capítulo 10, puede almacenar todos los tipos de datos de Python en archivos binarios, pero otros lenguajes no cuentan con módulos para cargar estos datos en sus programas.

En el resto del capítulo, representaré la siguiente estructura de datos de Python, que almacenan datos personales de una persona llamada Alice, en cada uno de estos formatos, para que puedas compararlos:

```
{  
    "name": "Alice Doe",  
    "age": 30,
```

```
“car”: None,  
  
“programmer”: True,  
  
“address”: {  
  
“street”: “100 Larkin St.”,  
  
“city”: “San Francisco”,  
  
“zip”: “94102”  
  
},  
  
“phone”: [  
  
{  
  
“type”: “mobile”,  
  
“number”: “415-555-7890”  
  
},  
  
{  
  
“type”: “work”,  
  
“number”: “415-555-1234”  
  
}  
  
]  
  
}
```

Estos formatos de texto tienen sus propias historias y su hueco en el ecosistema informático. Si tienes que elegir un formato de serialización para almacenar tus datos, ten en cuenta que JSON es más sencillo que XML y está más extendido que YAML, y que TOML se usa principalmente como formato para archivos de configuración. Por último, puede resultar tentador buscar tu propio formato de serialización de datos, pero es como reinventar la rueda

y tendrías que escribir también tu propio analizador para ese formato personalizado. Es mejor elegir un formato existente, sin más.

JSON

JSON almacena información como código fuente JavaScript, aunque lo usan muchas aplicaciones que no son de JavaScript. En particular, los sitios web suelen poner sus datos a disposición de programadores en formato JSON a través de API, como la API OpenWeather con la que trabajamos en el capítulo 13. Guardamos texto con formato JSON en archivos de texto simple con la extensión .json. Esta es la estructura de datos de ejemplo en formato de texto JSON:

```
{  
  "name": "Alice Doe",  
  "age": 30,  
  "car": null,  
  "programmer": true,  
  "address": {  
    "street": "100 Larkin St.",  
    "city": "San Francisco",  
    "zip": "94102"  
  },  
  "phone": [  
    {  
      "type": "mobile",  
      "number": "415-555-7890"
```

```
},  
  
{  
  
  "type": "work",  
  
  "number": "415-555-1234"  
  
}  
  
]  
  
}
```

Lo primero que notarás es que JSON se parece a la sintaxis de Python. Los diccionarios de Python y los objetos de JSON utilizan llaves y contienen pares clave-valor separados por comas, con cada clave y valor separados por dos puntos. Tanto las listas de Python como las matrices JSON utilizan corchetes y contienen valores separados por comas. En JSON, el espacio en blanco es insignificante fuera de cadenas con comillas dobles, lo que significa que puedes dejar espacio entre los valores como quieras. No obstante, es mejor dar formato anidado a objetos y matrices con sangría aumentada, como los bloques de código sangrados de Python. En nuestros datos de ejemplo, la lista de números de teléfono está sangrada dos espacios, con cada diccionario de número telefónico de la lista sangrado cuatro espacios.

Pero también hay diferencias entre JSON y Python. En lugar del valor None de Python, JSON utiliza la palabra clave null de JavaScript. Los valores booleanos son las palabras clave true y false de JavaScript, en minúsculas. JSON no admite comentarios de JavaScript ni cadenas multilínea; todas las cadenas de JSON deben emplear comillas dobles. A diferencia de las listas de Python, las matrices JSON no pueden tener comas finales, así que, aunque ["spam", "eggs"] es válido en JSON, ["spam", "eggs",] no lo es.

Facebook, Twitter, Yahoo!, Google, Tumblr, Wikipedia, Flickr, Data.gov, Reddit, IMDb, Rotten Tomatoes, LinkedIn y otros muchos sitios populares ofrecen API que trabajan con datos JSON. Algunos de estos sitios requieren registrarse, lo cual suele ser gratuito.

Tendrás que buscar documentación para descubrir qué URL necesita solicitar tu programa para obtener los datos que quieres, además del formato general de la estructura de datos JSON devuelta. Si el sitio que ofrece la API cuenta con una página para desarrolladores, busca ahí la documentación.

El módulo `json` de Python maneja los detalles de la traducción de una cadena formateada como datos JSON a los valores correspondientes de Python con las funciones `json.loads()` y `json.dumps()`. JSON no puede guardar todo tipo de valores de Python, solo aquellos de los siguientes tipos básicos de datos: cadenas, enteros, flotantes, booleanos, listas, diccionarios y `NoneType`. JSON no puede representar objetos específicos de Python, como `File`, objetos `reader` o `writer` de CSV, u objetos `WebElement` de Selenium. La documentación completa del módulo `json` se encuentra en <https://docs.python.org/3/library/json.html>.

Leer datos JSON

Para traducir una cadena que contiene datos JSON a un valor de Python, pásasela a la función `json.loads()`. Introduce lo siguiente en el intérprete de comandos interactivo:

```
1 > > > import json
```

```
> > > json_string = '{"name": "Alice Doe", "age": 30, "car":  
null, "programmer":
```

```
true, "address": {"street": "100 Larkin St.", "city": "San  
Francisco", "zip":
```

```
"94102"}, "phone": [{"type": "mobile", "number":  
"415-555-7890"}], {"type":
```

```
"work", "number": "415-555-1234"}]}'
```

```
2 > > > python_data = json.loads(json_string)
```

```
> > > python_data
```

```
{'name': 'Alice Doe', 'age': 30, 'car': None, 'programmer': True,
```

`'address':`

`{'street': '100 Larkin St.', 'city': 'San Francisco', 'zip': '94102'},`

`'phone': [{'type': 'mobile', 'number': '415-555-7890'}, {'type':
'work',`

`'number': '415-555-1234'}]}`

Después de importar el módulo `json` 1, puedes llamar a `loads()` 2 y pasarle una cadena de datos JSON. Observa que las cadenas JSON siempre llevan comillas dobles. Esto debería devolver los datos como un diccionario de Python.

Escribir datos JSON

La función `json.dumps()` convertirá datos de Python en una cadena de datos con formato JSON. Escribe esto en el intérprete de comandos interactivo:

```
> > > import json
```

```
> > > python_data = {'name': 'Alice Doe', 'age': 30, 'car':  
None, 'programmer': True, 'address':
```

```
{'street': '100 Larkin St.', 'city': 'San Francisco', 'zip': '94102'},  
'phone': [{'type':
```

```
'mobile', 'number': '415-555-7890'}, {'type': 'work', 'number':  
'415-555-1234'}]}
```

```
> > > json_string = json.dumps(python_data) 1
```

```
> > > print(json_string) 2
```

```
{"name": "Alice Doe", "age": 30, "car": null, "programmer": true,  
"address": {"street":
```

```
"100 Larkin St.", "city": "San Francisco", "zip": "94102"}, "phone":  
[{"type": "mobile",
```

```
"number": "415-555-7890"}, {"type": "work", "number":
```

```
"415-555-1234"}]}}
```

```
> > > json_string = json.dumps(python_data, indent=2) 3
```

```
> > > print(json_string)
```

```
{  
  "name": "Alice Doe",  
  "age": 30,  
  "car": null,  
  "programmer": true,  
  "address": {  
    "street": "100 Larkin St.",  
    "city": "San Francisco",  
    --fragmento omitido--  
  }  
}
```

El valor pasado a `json.dumps()` 1 puede pertenecer solo a estos tipos de datos de Python: cadenas, enteros, flotantes, booleanos, listas, diccionarios y `NoneType`.

Por defecto, todo el texto JSON está escrito en una sola línea 2. Este formato comprimido está bien para leer y escribir texto JSON entre programas, pero una forma multilínea sangrada será más fácil de leer para nosotros. El argumento `indent=2` 3 formatea el texto JSON en líneas separadas, con dos espacios de sangrado para cada diccionario o lista anidados. A menos que el texto JSON ocupe varios megabytes, merece la pena aumentar el tamaño añadiendo espacio y saltos de línea para favorecer la legibilidad.

Una vez que tenemos el texto JSON como un valor de cadena de Python, podemos escribirlo en un archivo `.json`, pasárselo a una función, usarlo en una solicitud web o realizar cualquier otra

operación que se pueda hacer con una cadena.

XML

El formato de archivo XML es más antiguo que JSON, pero se sigue utilizando mucho. Su sintaxis es similar a HTML, visto en el capítulo 18, e implica anidar etiquetas de apertura y cierre entre paréntesis angulares que contienen otro contenido. Estas etiquetas se denominan “elementos”. Los archivos de imagen SVG están hechos de texto escrito en XML. Los formatos de redifusión web RSS y Atom también están escritos en XML, y los documentos de Microsoft Word son simplemente archivos ZIP con extensión .docx que contienen archivos XML.

El texto con formato XML se guarda en archivos de texto simple con la extensión .xml. He aquí un ejemplo de estructura de datos formateada como XML:

```
<person >
```

```
<name > Alice Doe </name >
```

```
<age > 30 </age >
```

```
<programmer > true </programmer >
```

```
< car xsi:nil = "true" xmlns:xsi = "http://www.w3.org/2001/  
XMLSchema-instance" />
```

```
< address >
```

```
< street > 100 Larkin St. </street >
```

```
< city > San Francisco </city >
```

```
< zip > 94102 </zip >
```

```
< /address >
```

```
< phone >
```

```
< phoneEntry >
```


< type > mobile < /type >

< number > 415-555-7890 < /number >

< /phoneEntry >

< phoneEntry >

< type > work < /type >

< number > 415-555-1234 < /number >

< /phoneEntry >

< /phone >

< /person >

En este ejemplo, el elemento < person > tiene subelementos, como < name >, < age >, etc. Los subelementos < name > y < age > son elementos hijos y < person > es su elemento padre. Los documentos XML válidos deben tener un solo elemento raíz que contenga todos los demás elementos, como < person > en este ejemplo. Un documento con múltiples elementos raíz, como lo siguiente, no es válido:

< person > < name > Alice Doe < /name > < /person >

< person > < name > Bob Smith < /name > < /person >

< person > < name > Carol Watanabe < /name > < /person >

XML es bastante verboso en comparación con formatos de serialización más modernos, como JSON. Cada elemento tiene una etiqueta de apertura y cierre, como < age > y < /age >. Un elemento XML es un par clave-valor, siendo la clave la etiqueta del elemento (en este caso, < age >) y el valor el texto que va entre las etiquetas de apertura y cierre. El texto XML no tiene tipos de datos; todo lo que haya entre etiquetas de apertura y cierre se considera una cadena, incluidos los textos 94102 y true de nuestro ejemplo. Las listas de datos, como el elemento < phone >, tienen que nombrar sus elementos individuales con sus propios elementos,

como `<phoneEntry>`. El sufijo “Entry” para estos subelementos es solo una convención de nomenclatura.

Los comentarios en XML son idénticos a los de HTML: cualquier cosa entre `<!--` y `-->` va a ignorarse.

El espacio en blanco fuera de las etiquetas de apertura y cierre es insignificante y puede aplicársele el formato que se quiera. No existe el valor null en XML, pero podemos aproximarnos a eso añadiendo los atributos `xsi:nil = "true"` y `xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"` a una etiqueta. Los atributos XML son pares clave-valor escritos en formato `clave = "valor"` dentro de la etiqueta de apertura. La etiqueta se escribe como una etiqueta de autocierre; en vez de usar una etiqueta de cierre, la de apertura termina con `/>`, como en `<car xsi:nil = "true" xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance" />`.

Los nombres de atributos y etiquetas pueden escribirse en mayúsculas o en minúsculas, pero la convención es usar minúsculas. Los valores de los atributos pueden ir entre comillas simples o dobles, pero lo habitual son las dobles.

A veces no está claro si es mejor usar subelementos o atributos. Nuestro ejemplo emplea estos elementos para el dato de la dirección:

```
<address>
<street> 100 Larkin St. </street>
<city> San Francisco </city>
<zip> 94102 </zip>
</address>
```

Sin embargo, podríamos haber formateado fácilmente los datos del subelemento como atributos en un elemento `<address>` autocerrado:

```
<address street = "100 Larkin St." city = "San Francisco"
```

```
zip="94102" />
```

Este tipo de ambigüedades, junto con la naturaleza verbosa de las etiquetas, han hecho que XML vaya perdiendo popularidad. XML se utilizó mayormente durante las décadas de 1990 y 2000, y buena parte de ese software sigue actualmente en uso. Pero, a menos que tengas una razón concreta para utilizar XML, te va a ir mejor con JSON.

En general, las bibliotecas de software XML tienen dos formas de leer documentos XML. El enfoque DOM (*Document Object Model*, modelo de objetos de documento) lee todo el XML en memoria a la vez. De este modo, es más fácil acceder a los datos en cualquier parte del documento XML, pero, por lo general, solo funciona con archivos de tamaño pequeño o moderado. El enfoque SAX (*Simple API for XML*, API simple para XML) lee el documento XML como una secuencia de elementos, para no tener que cargar en memoria todo el documento a la vez. Este enfoque es ideal para archivos XML con varios gigabytes de tamaño, pero es menos cómodo porque no se puede trabajar con elementos hasta que se ha iterado por ellos en el documento.

La biblioteca estándar de Python cuenta con los módulos `xml.dom`, `xml.sax` y `xml.etree.ElementTree` para manejar texto XML. Para nuestros ejemplos sencillos, usaremos el módulo `xml.etree.ElementTree` para leer todo el documento XML entero de una vez.

Leer archivos XML

El módulo `xml.etree` utiliza objetos `Element` para representar un elemento XML y sus elementos hijos. Introduce lo siguiente en el intérprete de comandos interactivo:

```
1 >>> import xml.etree.ElementTree as ET
```

```
2 >>> xml_string = """<person> <name> Alice Doe </  
name> <age> 30 </age>
```

```
<programmer> true </programmer> <car xsi:nil="true"  
xmlns:xsi =
```

["http://www.w3.org/2001/XMLSchema-instance"/](http://www.w3.org/2001/XMLSchema-instance)

> <address> <street>

100 Larkin St. </street> <city> San Francisco </
city> <zip> 94102 </zip>

</address> <phone> <phoneEntry> <type> mobile </
type> <number> 415-555-

7890 </number> </
phoneEntry> <phoneEntry> <type> work </
type> <number>

415-555-1234 </number> </phoneEntry> </phone> </
person> """

3 > > > root = ET.fromstring(xml_string)

> > > root

<Element 'person' at 0x000001942999BBA0>

Importamos el módulo `xml.etree.ElementTree` 1 con la sintaxis `as ET`, de manera que podamos escribir `ET` en lugar del nombre completo del módulo `xml.etree.ElementTree`. La variable `xml_string` 2 contiene el texto del XML que queremos analizar, aunque este texto podría leerse igual de fácilmente desde un archivo de texto con la extensión `.xml`. Por último, pasamos este texto a la función `ET.fromstring()` 3, que devuelve un objeto `Element` que contiene los datos a los que queremos acceder. Guardaremos este objeto en una variable llamada `root`.

El módulo `xml.etree.ElementTree` también tiene una función `parse()`. Podemos pasarle el nombre de un archivo desde el que cargar XML y devuelve un objeto `Element`:

> > > import xml.etree.ElementTree as ET

> > > tree = ET.parse('my_data.xml')

> > > root = tree.getroot()

Una vez que tenemos un objeto `Element`, podemos acceder a sus atributos de Python `text` y `tag` para ver el nombre de la etiqueta y el texto que hay entre las etiquetas de apertura y cierre. Si pasamos el objeto `Element` a la función `list()`, debería devolver una lista con sus elementos hijos inmediatos. Continúa en el intérprete de comandos interactivos escribiendo esto:

```
> > > root.tag
```

```
'person'
```

```
> > > list(root)
```

```
[<Element 'name' at 0x00000150BA4ADDF0>, <Element 'age' at
```

```
0x00000150BA4ADF30>, <Element 'programmer' at
0x00000150BA4ADEE0> ,
```

```
<Element 'car' at 0x00000150BA4ADD00>, <Element 'address' at
```

```
0x00000150BA4ADCB0>, <Element 'phone' at
0x00000150BA4ADA30>]
```

Se puede acceder a los objetos `Element` hijos de un objeto `Element` padre mediante un índice entero, como sucede con las listas de Python. Así pues, si `root` contiene el elemento `<person>`, `root[0]` y `root[1]` contienen los elementos `<name>` y `<age>`, respectivamente. Podemos acceder a los atributos `tag` y `text` de todos estos objetos `Element`. Sin embargo, cualquier etiqueta autocerrada, como `<car/>`, usará `None` como atributo de texto. Por ejemplo, escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > root[0].tag
```

```
'name'
```

```
> > > root[0].text
```

```
'Alice Doe'
```

```
> > > root[3].tag
```

`'car'`

```
> > > root[3].text = None # <car/> no tiene texto.
```

True

```
> > > root[4].tag
```

`'address'`

```
> > > root[4][0].tag
```

`'street'`

```
> > > root[4][0].text
```

`'100 Larkin St.'`

Desde el elemento root, podemos explorar los datos de todo el documento XML. También podemos iterar por los elementos hijos inmediatos poniendo un objeto Element dentro de un bucle for:

```
> > > for elem in root:
```

```
... print(elem.tag, , elem.text)
```

```
...
```

```
name -- Alice Doe
```

```
age -- 30
```

```
programmer -- true
```

```
car -- None
```

```
address -- None
```

```
phone -- None
```

Si quieres iterar por todos los hijos del objeto Element, puedes llamar al método `iter()` en un bucle for:

```
> > > for elem in root.iter():
```

```
... print(elem.tag, , elem.text)
```

```
.
```

```
person -- None
```

```
name -- Alice Doe
```

```
age -- 30
```

```
programmer -- true
```

```
car -- None
```

```
address -- None
```

```
street -- 100 Larkin St.
```

```
city -- San Francisco
```

```
zip -- 94102
```

```
phone -- None
```

```
phoneEntry -- None
```

```
type -- mobile
```

```
number -- 415-555-7890
```

```
phoneEntry -- None
```

```
type -- work
```

```
number -- 415-555-1234
```

De manera opcional, podemos pasar una cadena al método `iter()` para filtrar elementos XML con una etiqueta coincidente. Este ejemplo llama a `iter('number')` para iterar solo por los elementos hijos `<number>` del elemento raíz:

```
> > > for elem in root.iter('number'):
```

```
... print(elem.tag, , elem.text)
```

```
.
```

```
number -- 415-555-7890
```

```
number -- 415-555-1234
```

La navegación por los datos de un documento XML abarca mucho más que los atributos y métodos tratados en este apartado. Por ejemplo, igual que los selectores CSS que vimos en el capítulo 13 pueden buscar elementos en el HTML de una página web, un lenguaje llamado XPath puede localizar elementos en un documento XML. Estos conceptos quedan fuera del alcance de este libro, pero puedes aprender más sobre ellos en la documentación de Python: <https://docs.python.org/3/library/xml.etree.elementtree.html>.

Los módulos XML de Python no tienen forma de convertir texto XML en una estructura de datos de Python. No obstante, podemos hacerlo con el módulo de terceros `xmldict` (<https://pypi.org/project/xmldict/>). Tienes las instrucciones de instalación completas en el apéndice A. Aquí vemos un ejemplo de uso:

```
> > > import xmldict
```

```
> > > xml_string = """<person> <name> Alice Doe </  
name> <age> 30 </age>
```

```
<programmer> true </programmer> <car xsi:nil = "true"  
xmlns:xsi =
```

```
"http://www.w3.org/2001/XMLSchema-instance"/
```

```
> <address> <street>
```

```
100 Larkin St. </street> <city> San Francisco </  
city> <zip> 94102
```

```
</zip> </
```

```
address> <phone> <phoneEntry> <type> mobile </  
type> <number>
```



```

415-555-7890 </number> </
phoneEntry> <phoneEntry> <type> work </type>

<number> 415-555-1234 </number> </phoneEntry> </
phone> </person> """

> > > python_data = xmltodict.parse(xml_string)

> > > python_data

{'person': {'name': 'Alice Doe', 'age': '30', 'programmer': 'true',
'car': {'@xsi:nil': 'true', '@xmlns:xsi': 'http://www.w3.org/2001/
XMLSchema-instance'}, 'address': {'street': '100 Larkin St.', 'city':
'San Francisco', 'zip': '94102'}, 'phone': {'phoneEntry': [{'type':
'mobile', 'number': '415-555-7890'}, {'type': 'work', 'number':
'415-555-1234'}]}}}

```

Una de las razones por las que el estándar XML se va quedando atrás respecto a formatos como JSON es que representar tipos de datos en XML es mucho más complicado. Por ejemplo, el elemento `<programmer>` se ha analizado como el valor de cadena `'true'` en lugar de como el valor booleano `True`. Y el elemento `<car>` se ha analizado como el extraño par clave-valor `'car': {'@xsi:nil': 'true', '@xmlns:xsi': 'http://www.w3.org/2001/XMLSchema-instance'}` y no como el valor `None`. Hay que comprobar siempre la entrada y salida de cualquier módulo XML para asegurarse de que está representando los datos como queríamos.

Escribir archivos XML

El módulo `xml.etree` es un poco difícil de manejar, así que, para proyectos pequeños, es mejor llamar a la función `open()` y el método `write()` para crear texto XML nosotros mismos. Pero, para crear un documento XML desde cero con el módulo `xml.etree`, necesitaremos crear un objeto `Element` raíz (como `<person>` en nuestro ejemplo) y llamar después a la función `SubElement()` para

crearle elementos hijos. Se puede configurar cualquier atributo XML del elemento con el método `set()`. Por ejemplo, escribe lo siguiente:

```
> > > import xml.etree.ElementTree as ET

> > > person = ET.Element('person') # Crea el elemento XML raíz.

> > > name = ET.SubElement(person, 'name') # Crea
<name> y lo pone en <person>.

> > > name.text = 'Alice Doe' # Establece el texto entre
<name> y </name>.

> > > age = ET.SubElement(person, 'age')

> > > age.text = '30' # El contenido XML siempre es una cadena.

> > > programmer = ET.SubElement(person, 'programmer')

> > > programmer.text = 'true'

> > > car = ET.SubElement(person, 'car')

> > > car.set('xsi:nil', 'true')

> > > car.set('xmlns:xsi', 'http://www.w3.org/2001/
XMLSchema-instance')

> > > address = ET.SubElement(person, 'address')

> > > street = ET.SubElement(address, 'street')

> > > street.text = '100 Larkin St.'
```

Para mayor brevedad, dejaré el resto de elementos `<address>` y `<phone>`. Llama a las funciones `ET.tostring()` y `decode()` con el objeto `Element` raíz para obtener una cadena de Python del texto XML:

```
> > > ET.tostring(person, encoding='utf-8').decode('utf-8')

'<person> <name> Alice Doe </name> <age> 30 </
```

```
age > <programmer > true </programmer >
```

```
<car xsi:nil="true" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance"/>
```

```
<address > <street > 100 Larkin St. </street > </address > </
person >'
```

Es una pena que la función `tostring()` devuelva un objeto bytes en vez de una cadena, lo que requiere una llamada al método `decode()` para obtener una cadena real. Pero, cuando ya tenemos el texto XML como valor de cadena de Python, podemos escribirlo en un archivo .xml, pasárselo a una función, usarlo en una solicitud web o hacer cualquier otra cosa que se pueda hacer con una cadena.

Resumen

CSV, JSON y XML son formatos de texto simple habituales para guardar datos. Son fáciles de analizar para los programas, al tiempo que siguen siendo legibles para los humanos, así que se utilizan a menudo para datos de hojas de cálculo sencillas o web. Los módulos `csv`, `json` y `xml.etree.ElementTree` de la biblioteca estándar de Python simplifican enormemente el proceso de leer y escribir estos archivos, así que no es necesario hacerlo con la función `open()`.

Estos formatos no son específicos de Python; los usan también muchos otros lenguajes de programación y aplicaciones de software. Este capítulo puede ser útil para escribir programas de Python que también interactúen con cualquier aplicación que los use.

Preguntas prácticas

1. ¿Cuáles son algunas de las hojas de cálculo de Excel que no tienen las CSV?
2. ¿Qué hay que pasar a `csv.reader()` y `csv.writer()` para crear objetos reader y writer?
3. ¿En qué modos hay que abrir los objetos File para objetos reader y writer?

- 4.¿Qué método toma un argumento de lista y lo escribe en un archivo CSV?
- 5.¿Qué hacen los argumentos delimiter e lineterminator?
- 6.Entre CSV, JSON y XML, ¿qué formatos se pueden editar fácilmente con un editor de texto?
- 7.¿Qué función toma una cadena de datos JSON y devuelve una estructura de datos de Python?
- 8.¿Qué función toma una estructura de datos de Python y devuelve una cadena de datos JSON?
- 9.¿Qué formato de serialización de datos se parece a HTML, con etiquetas entre paréntesis angulares?
- 10.¿Cómo escribe JSON valores None?
- 11.¿Cómo escribimos valores booleanos en JSON?

Programa de práctica: convertidor de Excel a CSV

Excel puede guardar una hoja de cálculo en un archivo CSV con unos pocos clics del ratón, pero, si tuviéramos que convertir cientos de archivos Excel en CSV, estaríamos hablando de horas haciendo clic. Utilizando el módulo `openpyxl` del capítulo 14, escribe un programa que lea todos los archivos Excel que haya en el directorio de trabajo actual y los convierta en archivos CSV.

Un solo archivo Excel puede contener varias hojas; tendrás que crear un CSV por hoja. Los nombres de los archivos CSV deberían ser `<nombreexcel>_<nombrehoja>.csv`, donde `<nombreexcel>` es el nombre del Excel sin la extensión de archivo (por ejemplo, `spam_data`, no `spam_data.xlsx`) y `<nombrehoja>` es la cadena de la variable `title` del objeto `Worksheet`. Este programa implicará muchos bucles `for` anidados. El esqueleto del programa sería algo así:

```
for excel_file in os.listdir('.'):
```

Se salta los archivos que no son xlsx, carga el objeto de la hoja de cálculo.

for sheet_name in wb.sheetnames:

Pasa en bucle por todas las hojas del archivo.

Crea el nombre de archivo del CSV a partir del nombre del Excel y el de la hoja.

Crea el objeto csv.writer para este archivo CSV.

Pasa en bucle por todas las celdas de la hoja.

for row_num in range(1, sheet.max_row + 1):

row_data = [] # Añade cada celda a esta lista.

Pasa en bucle por todas las celdas de la fila.

for col_num in range(1, sheet.max_column + 1):

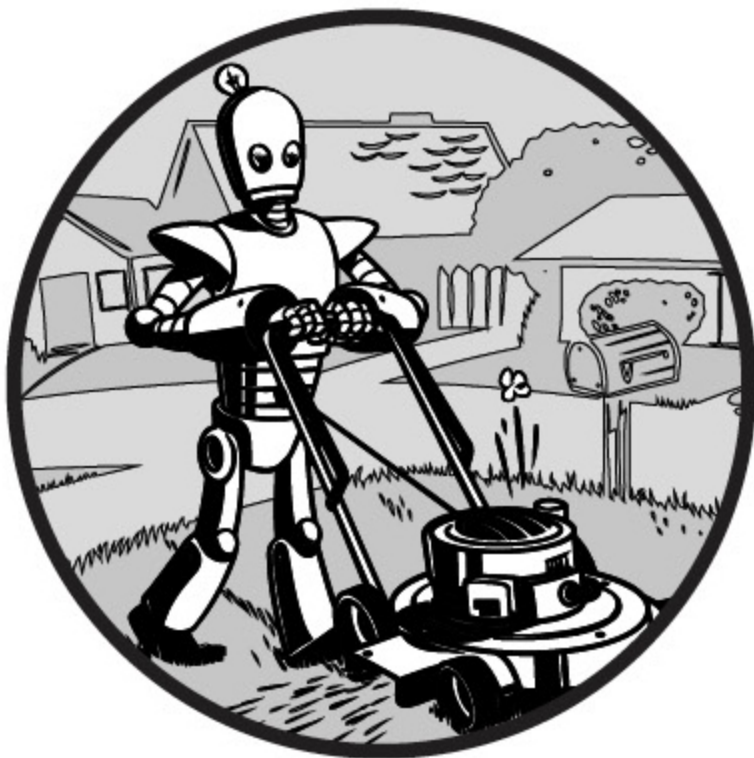
Añade los datos de cada celda a row_data.

Escribe la lista row_data en el archivo CSV.

csv_file.close()

Descarga el archivo excelSpreadsheets.zip de los recursos en línea del libro y descomprime las hojas de cálculo en la misma carpeta de tu programa. Puedes usar estos archivos para probar el programa.

Manejar el tiempo, programar tareas y lanzar programas



Ejecutar programas mientras estás sentado frente al ordenador está bien, pero también es útil que los programas se ejecuten sin tu supervisión directa. El reloj del ordenador puede programar la ejecución de código en una fecha y hora específicas o a intervalos regulares. Por ejemplo, nuestro programa podría rastrear un sitio web cada hora para comprobar si hay cambios o realizar una tarea que requiera un uso intensivo de la CPU a las 4 de la madrugada mientras dormimos. Los módulos `time` y `datetime` de Python ofrecen estas funciones.

También podemos escribir programas que ejecuten otros programas

según una programación utilizando el módulo subprocess. A menudo, la forma más rápida de programar es aprovechar las aplicaciones que otras personas ya han escrito.

El módulo time

El reloj del sistema de nuestro ordenador está configurado con una fecha, hora y zona horaria específicas. El módulo time integrado permite a nuestros programas de Python leer el reloj del sistema para conocer la hora actual. Las funciones más útiles de este módulo son time.time(), que devuelve un valor denominado “marca de tiempo *epoch*”, y time.sleep(), que pausa un programa.

Devolver la marca de tiempo epoch

El *epoch* de Unix es una referencia temporal que se utiliza habitualmente en programación: la medianoche del 1 de enero de 1970, hora universal coordinada (UTC). La función time.time() devuelve el número de segundos transcurridos desde ese momento como un valor flotante. (Recuerda que un flotante es simplemente un número con un punto decimal). Este número se denomina marca de tiempo *epoch*. Para ver un ejemplo, introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > import time
```

```
> > > time.time()
```

```
1773813875.3518236
```

Aquí, estoy llamando a time.time() con la fecha 17 de marzo de 2026 y la hora 11:04 pm, hora estándar del Pacífico. El valor de retorno son los segundos transcurridos entre el *epoch* de Unix y el momento en el que se ha llamado a time.time().

El valor de retorno de time.time() es útil, pero ilegible para los humanos. La función time.ctime() devuelve una descripción de la hora actual. Como opción, podemos pasar el número de segundos desde el *epoch* de Unix, devuelto por time.time(), para obtener un valor de cadena de esa hora. Escribe esto en el intérprete de

comandos interactivo:

```
> > > import time
```

```
> > > time.ctime()
```

```
'Tue Mar 17 11:05:38 2026'
```

```
> > > this_moment = time.time()
```

```
> > > time.ctime(this_moment)
```

```
'Tue Mar 17 11:05:45 2026'
```

Las marcas de tiempo *epoch* sirven para perfilar código, es decir, medir cuánto tarda en ejecutarse un fragmento de código. Si llamamos a `time.time()` al principio del bloque de código que queremos cronometrar y lo hacemos otra vez al final, podemos restar la primera marca de tiempo de la segunda para calcular los segundos transcurridos entre ambas llamadas. Por ejemplo, abre una nueva pestaña en el editor de archivos y escribe el siguiente programa:

```
# Mide lo que se tarda en multiplicar 100 000 números.
```

```
import time
```

```
1 def calculate_product():
```

```
# Calcula el producto de los primeros 100 000 números.
```

```
product = 1
```

```
for i in range(1, 100001):
```

```
product = product * i
```

```
return product
```

```
2 start_time = time.time()
```

```
result = calculate_product()
```



```
3 end_time = time.time()
```

```
4 print(f'It took {end_time - start_time} seconds to calculate.')
```

En 1, definimos una función `calculate_product()` para pasar en bucle por los enteros de 1 a 100.000 y devolver su producto. En 2, llamamos a `time.time()` y lo guardamos en `start_time`. Justo después de llamar a `calculate_product()`, llamamos a `time.time()` otra vez y lo guardamos en `end_time`. Terminamos imprimiendo cuánto ha tardado en ejecutarse `calculate_product()`.

Guarda este programa como `calcProd.py` y ejecútalo. La salida será similar a esto:

```
It took 2.844162940979004 seconds to calculate.
```

Otra manera de perfilar el código es usar la función `cProfile.run()`, que ofrece un nivel informativo más detallado que la sencilla técnica de `time.time()`. Puedes leer más sobre la función `cProfile.run()` en el capítulo 13 de mi otro libro, *Beyond the Basic Stuff with Python* (No Starch Press, 2020).

Pausar programas

Si necesitas pausar un programa un momento, llama a la función `time.sleep()` y pásale el número de segundos que quieres que el programa permanezca en pausa. Por ejemplo, introduce esto en el intérprete de comandos interactivo:

```
> > > import time
```

```
> > > for i in range(3):
```

```
... 1 print('Tick')
```

```
... 2 time.sleep(1)
```

```
... 3 print('Tock')
```

```
... 4 time.sleep(1)
```

```
.
```

Tick

Tock

Tick

Tock

Tick

Tock

```
5 > > > time.sleep(5)
```

```
> > >
```

El bucle for imprimirá Tick 1, hará una pausa de un segundo 2, imprimirá Tock 3, hará una pausa de un segundo 4, imprimirá Tick, hará la pausa y así sucesivamente hasta que Tick y Tock se hayan imprimido tres veces cada uno.

La función `time.sleep()` se bloqueará (es decir, no volverá ni dejará que el programa ejecute otro código) hasta que haya transcurrido el número de segundos que le hemos pasado. Por ejemplo, si introducimos `time.sleep(5)` 5, veremos que el siguiente *prompt* (`> > >`) no aparece hasta que no hayan pasado cinco segundos.

Proyecto 14: Supercronómetro

Supongamos que quieres conocer el tiempo que dedicas a tareas aburridas que todavía no has automatizado. No tienes a mano un cronómetro físico y es sorprendentemente difícil encontrar una *app* de cronómetro gratuita para portátil o *smartphone* que no esté llena de anuncios y no mande una copia de tu historial de navegación a los anunciantes. (Dice que puede hacerlo en los términos y condiciones que has aceptado. Los has leído, ¿no?). Puedes escribir un sencillo programa de cronómetro en Python.

Esto es lo que hará el programa:

- Descubrir la hora actual llamando a `time.time()` y guardarla como marca de tiempo al principio del programa y al principio de cada

vuelta.

- Llevar un contador de vueltas e incrementarlo cada vez que se pulse **Intro**.
- Calcular el tiempo transcurrido restando marcas de tiempo.
- Manejar la excepción `KeyboardInterrupt` para que el usuario pueda pulsar **Control-C** para salir.

Abre una nueva pestaña en el editor de archivos y guárdala como `stopwatch.py`.

Paso 1: configurar el programa para que rastree las horas

El cronómetro tendrá que usar la hora actual, así que necesitas importar el módulo `time`. El programa también debería imprimir unas instrucciones breves para el usuario antes de llamar a `input()` para que el tiempo empiece a correr después de pulsar **Intro**. A continuación, el código empezará a rastrear las vueltas cada vez que el usuario pulse **Intro** hasta que pulse **Control-C** para salir. Escribe el siguiente código en el editor de archivos, usando un comentario `PENDIENTE` como marcador para el resto del código:

```
# Un sencillo programa de cronómetro
```

```
import time
```

```
# Muestra las instrucciones del programa.
```

```
print('Press ENTER to begin and to mark laps. Ctrl-C quits.')
```

```
input() # Pulsa Intro para comenzar.
```

```
print('Started.')
```

```
start_time = time.time() # Obtiene la hora de inicio de la primera vuelta.
```

```
last_time = start_time
```

```
lap_number = 1
```

PENDIENTE: Empezar a rastrear los tiempos de las vueltas.

Ahora que ya has escrito el código para mostrar las instrucciones, empieza la primera vuelta, anota el tiempo y establece lap_number en 1.

Paso 2: rastrear e imprimir los tiempos de las vueltas

Vamos a escribir el código para empezar cada vuelta nueva, calcular cuánto ha tardado la anterior y calcular el tiempo total transcurrido desde que se inició el cronómetro.

Mostraremos el tiempo de la vuelta y el tiempo total y aumentaremos el contador de vueltas con cada nueva vuelta. Añade este código a tu programa:

Un sencillo programa de cronómetro

import time

--fragmento omitido--

Empieza a rastrear los tiempos de las vueltas.

1 try:

2 while True:

input()

3 lap_time = round(time.time() - last_time, 2)

4 total_time = round(time.time() - start_time, 2)

5 print('Lap #{lap_number}: {total_time}({lap_time})', end = '')

lap_number = 1

last_time = time.time() # Reinicia el tiempo de la última vuelta.

6 except KeyboardInterrupt:

Maneja la excepción Control-C para evitar que se muestre su mensaje de error.

```
print('\nDone.')
```

Si el usuario pulsa **Control-C** para detener el cronómetro, se lanzará la excepción `KeyboardInterrupt` y el programa fallará. Para evitar el fallo, envolvemos esta parte del programa en una declaración `try` 1. Manejaremos la excepción en la proposición `except` 6, que imprime `Done` cuando se lance la excepción en lugar de mostrar el mensaje de error de `KeyboardInterrupt`. Hasta que eso ocurra, la ejecución se produce dentro de un bucle infinito 2 que llama a `input()` y espera a que el usuario pulse **Intro** para terminar una vuelta. Cuando termina una vuelta, calculamos lo que ha durado restando la hora de inicio de la vuelta, `last_time`, a la hora actual, `time.time()` 3. Calculamos el tiempo total transcurrido restando la hora de inicio del cronómetro, `start_time`, a la hora actual 4. Como el resultado de estos cálculos tendrá muchos dígitos detrás del punto decimal (como 4.766272783279419), usamos la función `round()` para redondear el valor flotante a dos dígitos en 3 y 4.

En 5, imprimimos el número de vuelta, el tiempo total transcurrido y el tiempo de la vuelta. Como el hecho de que el usuario pulse **Intro** para la llamada a `input()` imprimirá una nueva línea en la pantalla, pasa `end=""` a la función `print()` para evitar dobles espacios en la salida. Tras imprimir la información de la vuelta, nos preparamos para la siguiente vuelta añadiendo 1 al contador `lap_number` y estableciendo `last_time` en la hora actual, que será la hora de inicio de la siguiente vuelta.

Ideas para programas similares

El seguimiento del tiempo abre varias posibilidades para tus programas. Aunque puedes descargar aplicaciones para hacer algunas de estas cosas, la ventaja de escribir programas tú mismo es que serán gratuitos y no estarán llenos de anuncios y funciones inútiles. Podrías escribir programas similares para hacer lo siguiente:

- Crear una sencilla aplicación de hoja de tiempo que registre

cuándo escribes el nombre de una persona y utilice la hora actual para registrar su entrada o salida.

- Añadir una función al programa para mostrar el tiempo transcurrido desde que se inició un proceso, como una descarga que utiliza el módulo `requests` (véase el capítulo 13).

- Comprobar de forma intermitente cuánto tiempo lleva ejecutándose un programa y ofrecer al usuario la posibilidad de cancelar las tareas que están tardando demasiado.

El módulo `datetime`

El módulo `time` es útil para obtener una marca de tiempo *epoch* de Unix con la que trabajar, pero, si quieres mostrar una fecha en un formato más cómodo o hacer cuentas con fechas (por ejemplo, averiguar qué fecha era hace 205 días o qué fecha será dentro de 123), deberías usar el módulo `datetime`. El módulo `datetime` tiene su propio tipo de datos `datetime`. Los valores `datetime` representan un momento específico. Introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > import datetime
```

```
1 > > > datetime.datetime.now()
```

```
2 datetime.datetime(2026, 2, 27, 11, 10, 49, 727297)
```

```
3 > > > dt = datetime.datetime(2026, 10, 21, 16, 29, 0)
```

```
4 > > > dt.year, dt.month, dt.day
```

```
(2026, 10, 21)
```

```
5 > > > dt.hour, dt.minute, dt.second
```

```
(16, 29, 0)
```

Llamar a `datetime.datetime.now()` 1 devuelve un objeto `datetime` 2 para la fecha y la hora actuales, según el reloj de tu ordenador. Este objeto incluye el año, el mes, el día, la hora, el segundo y el microsegundo del momento actual. También se puede recuperar un

objeto `datetime` para un momento específico con la función `datetime.datetime()`3, pasándole enteros que representen el año, mes, día, hora y segundo del momento deseado. Estos enteros se guardarán en los atributos `year`, `month`, `day` 4, `hour`, `minute` y `second` 5 del objeto.

Una marca de tiempo *epoch* de Unix se puede convertir en un objeto `datetime` con la función `datetime.datetime.fromtimestamp()`. La fecha y la hora del objeto `datetime` se convertirá acorde a la zona horaria local. Introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > import datetime, time
```

```
> > > datetime.datetime.fromtimestamp(1000000)
```

```
datetime.datetime(1970, 1, 12, 5, 46, 40)
```

```
> > > datetime.datetime.fromtimestamp(time.time())
```

```
datetime.datetime(2026, 10, 21, 16, 30, 0, 604980)
```

Llamar a `datetime.datetime.fromtimestamp()` y pasarle 1000000 devuelve un objeto `datetime` para el momento un millón de segundos después del *epoch* de Unix. Si le pasamos `time.time()`, la marca de tiempo *epoch* de Unix para el momento actual, devuelve un objeto `datetime` para el momento actual. Así pues, las expresiones `datetime.datetime.now()` y `datetime.datetime.fromtimestamp(time.time())` hacen lo mismo, ambas nos dan un objeto `datetime` para el momento presente.

Podemos comparar objetos `datetime` con operadores de comparación para ver cuál precede al otro. El último objeto `datetime` es el valor “mayor”. Escribe esto en el intérprete de comandos interactivo:

```
> > > import datetime
```

```
1 > > > halloween_2026 = datetime.datetime(2026, 10, 31, 0, 0, 0)
```

```
2 > > > new_years_2027 = datetime.datetime(2027, 1, 1, 0, 0, 0,
```

0)

```
> > > oct_31_2026 = datetime.datetime(2026, 10, 31, 0, 0, 0)
```

```
3 > > > halloween_2026 = oct_31_2026
```

True

```
4 > > > halloween_2026 > new_years_2027
```

False

```
5 > > > new_years_2027 > halloween_2026
```

True

```
> > > new_years_2027 = oct_31_2026
```

True

Este código crea un objeto datetime para el primer momento (medianoche) del 31 de octubre de 2026 y lo guarda en halloween_2026 1. A continuación, crea un objeto datetime para el primer momento del 1 de enero de 2027 y lo guarda en new_years_2027 2. Crea otro objeto para la medianoche del 31 de octubre de 2026 y lo guarda en oct_31_2026. Comparar halloween_2026 y oct_31_2026 muestra que son iguales 3. Comparar new_years_2027 y halloween_2026 muestra que new_years_2027 es mayor (posterior) que halloween_2026 4 5.

Representar la duración

El módulo datetime también ofrece un tipo de datos timedelta, que representa una duración en vez de un momento. Introduce lo siguiente en el intérprete de comandos:

```
> > > import datetime
```

```
1 > > > delta = datetime.timedelta(days=11, hours=10,  
minutes=9, seconds=8)
```

```
2 > > > delta.days, delta.seconds, delta.microseconds
```



```
(11, 36548, 0)
```

```
> > > delta.total_seconds()
```

```
986948.0
```

```
> > > str(delta)
```

```
'11 days, 10:09:08'
```

Para crear un objeto `timedelta`, usa la función `datetime.timedelta()`. La función `datetime.timedelta()` toma los argumentos `weeks`, `days`, `hours`, `minutes`, `seconds`, `milliseconds` y `microseconds`. No hay argumento `month` ni `year` porque un año y un mes son una cantidad de tiempo variable, dependiendo del mes y el año concretos. Un objeto `timedelta` tiene la duración total representada en días, segundos y microsegundos. Estos números se almacenan en los atributos `days`, `seconds` y `microseconds`, respectivamente. El método `total_seconds()` devolverá la duración solo en segundos. Pasar un objeto `timedelta` a `str()` devolverá una representación del objeto en una cadena con un formato agradable y legible para los humanos.

En este ejemplo, pasamos argumentos de palabra clave a `datetime.timedelta()` para especificar una duración de 11 días, 10 horas, 9 minutos y 8 segundos, y guardamos el objeto `timedelta` devuelto en `delta` 1. El atributo `days` de este objeto `timedelta` almacena 11, y su atributo `seconds` almacena 36548 (10 horas, 9 minutos y 8 segundos expresadas en segundos) 2. Llamar a `total_seconds()` nos dice que 11 días, 10 horas, 9 minutos y 8 segundos son 986 948 segundos. Por último, pasar el objeto `timedelta` a `str()` devuelve una cadena que describe la duración de manera sencilla.

Podemos usar los operadores aritméticos para hacer cálculos de fecha con valores `datetime`. Por ejemplo, para calcular la fecha que será 1.000 días a partir de hoy, escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > import datetime
```

```
> > > now = datetime.datetime.now()
```

```
> > > now
```

```
datetime.datetime(2026, 12, 2, 18, 38, 50, 636181)
```

```
> > > thousand_days = datetime.timedelta(days=1000)
```

```
> > > now + thousand_days
```

```
datetime.datetime(2029, 8, 28, 18, 38, 50, 636181)
```

Primero, creamos un objeto datetime para el momento actual y lo almacenamos en now. Después, creamos un objeto timedelta para una duración de 1.000 días y lo guardamos en thousand_days. Si sumamos now y thousand_days, obtenemos un objeto datetime para la fecha de 1.000 días contando desde la fecha y hora actuales. Python calculará la fecha para saber que mil días después del 2 de diciembre de 2026 será el 28 de agosto de 2029. Al calcular mil días a partir de una fecha dada, hay que recordar cuántos días tiene cada mes y tener en cuenta los años bisiestos y otros detalles delicados. El módulo datetime se ocupa de eso por nosotros.

Podemos sumar o restar objetos timedelta con objetos datetime o con otros objetos timedelta utilizando los operadores + y -. Un objeto timedelta se puede multiplicar o dividir por valores enteros o flotantes con los operadores * y /. Escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > import datetime
```

```
1 > > > oct_21st = datetime.datetime(2026, 10, 21, 16, 29, 0)
```

```
2 > > > about_thirty_years = datetime.timedelta(days=365 * 30)
```

```
> > > oct_21st
```

```
datetime.datetime(2026, 10, 21, 16, 29)
```

```
> > > oct_21st - about_thirty_years
```

```
datetime.datetime(1996, 10, 28, 16, 29)
```

```
> > > oct_21st - (2 * about_thirty_years)
```

```
datetime.datetime(1966, 11, 5, 16, 29)
```

Aquí, creamos un objeto `datetime` para el 21 de octubre de 2026 1 y un objeto `timedelta` para una duración de unos 30 años 2. (Estamos usando 365 días por año, ignorando los bisiestos). Restar `about_thirty_years` de `oct_21st` nos da un objeto `datetime` para la fecha 30 años antes del 21 de octubre de 2026. Restar 2 * `about_thirty_years` de `oct_21st` devuelve un objeto `datetime` para la fecha unos 60 años antes: la primera hora de la tarde del 5 de noviembre de 1966.

Pausar hasta una fecha específica

El método `time.sleep()` permite poner un programa en pausa durante un número de segundos determinado. Usando un bucle `while`, podemos pausar los programas hasta una fecha específica. Por ejemplo, este código continuará hasta Halloween de 2039:

```
import datetime
```

```
import time
```

```
halloween_2039 = datetime.datetime(2039, 10, 31, 0, 0, 0)
```

```
while datetime.datetime.now() < halloween_2039:
```

```
    time.sleep(1) # Espera 1 segundo antes de seguir con el bucle de comprobación.
```

La llamada a `time.sleep(1)` detendrá el programa de Python para que el ordenador no malgaste ciclos de procesamiento de CPU comprobando la hora una y otra vez lo más rápido posible. En su lugar, el bucle `while` comprobará la condición una vez por segundo y seguirá con el resto del programa después de Halloween de 2039.

Convertir objetos `datetime` en cadenas

Las marcas de tiempo *epoch* y los objetos `datetime` no son muy agradables a la vista humana. Utiliza el método `strftime()` para mostrar un objeto `datetime` como una cadena. (La *f* del nombre de la función `strftime()` es de formato).

El método `strftime()` utiliza directivas similares al formato de cadenas de Python. La tabla 19.1 recoge una lista completa de directivas de `strftime()`. También puedes consultar el sitio web <https://strftime.org> para más información.

Tabla 19.1. Directivas de `strftime()`.

Directiva	Significado
<code>%Y</code>	Año con siglo, como '2026'.
<code>%y</code>	Año sin siglo, de '00' a '99' (de 1970 a 2069).
<code>%M</code>	Mes como número decimal, de '01' a '12'.
<code>%B</code>	Nombre completo del mes, como 'November'.
<code>%b</code>	Nombre del mes abreviado, como 'Nov'.
<code>%d</code>	Día del mes, de '01' a '31'.
<code>%j</code>	Día del año, de '001' a '366'.
<code>%w</code>	Día de la semana, de '0' (domingo) a '6' (sábado).
<code>%A</code>	Nombre completo del día de la semana, como 'Monday'.
<code>%a</code>	Nombre abreviado del día de la semana, como 'Mon'.
<code>%H</code>	Hora (24 horas), de '00' a '23'.
<code>%h</code>	Hora (12 horas), de '01' a '12'.
<code>%M</code>	Minuto, de '00' a '59'.
<code>%S</code>	Segundo, de '00' a '59'.
<code>%p</code>	'AM' o 'PM'.
<code>%Z</code>	Carácter '%' literal.

Pasa a `strftime()` una cadena con formato personalizado que contenga directivas de formateo (junto con barras, dos puntos, etc.), y la función devolverá la información del objeto `datetime` como cadena con formato. Introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > oct_21st = datetime.datetime(2026, 10, 21, 16, 29, 0)
```

```
> > > oct_21st.strftime("%Y/%m/%d %H:%M:%S")
```

```
'2026/10/21 16:29:00'
```

```
> > > oct_21st.strftime("%I:%M %p")
```

```
'04:29 PM'
```

```
> > > oct_21st.strptime("%B of %y")
```

```
"October of '26"
```

Aquí, tenemos un objeto `datetime` para el 21 de octubre de 2026 a las 4:29 pm, guardado en `oct_21st`. Pasar la cadena de formato personalizado `'%Y/%m/%d %H:%M:%S'` a `strptime()` devuelve una cadena que contiene 2026, 10 y 21 separados por barras oblicuas, y 16, 29 y 00 separados por dos puntos. Pasar `'%I:%M %p'` devuelve `'04:29 PM'`, y pasar `"%B of %y"` devuelve `"October of '26"`.

Convertir cadenas en objetos `datetime`

Si tienes una cadena con información de fecha, como `'2026/10/21 16:29:00'` o `'October 21, 2026'`, y necesitas transformarla en un objeto `datetime`, utiliza la función `datetime.datetime.strptime()`. La función `strptime()` es la contraria del método `strftime()` y hay que pasarle una cadena con formato personalizado usando las mismas directivas que en `strftime()` para que la función sepa cómo analizar y comprender la cadena. Introduce lo siguiente en el intérprete de comandos interactivo:

```
1 > > > datetime.datetime.strptime('October 21, 2026', '%B
%d, %Y')
```

```
datetime.datetime(2026, 10, 21, 0, 0)
```

```
> > > datetime.datetime.strptime('2026/10/21 16:29:00',
'%Y/%m/%d %H:%M:%S')
```

```
datetime.datetime(2026, 10, 21, 16, 29)
```

```
> > > datetime.datetime.strptime("October of '26", "%B of
'%y")
```

```
datetime.datetime(2026, 10, 1, 0, 0)
```

```
> > > datetime.datetime.strptime("November of '63", "%B of
'%y")
```

```
2 datetime.datetime(2063, 11, 1, 0, 0)
```

```
> > > datetime.datetime.strptime("November of '73", "%B of '%y")
```

```
3 datetime.datetime(1973, 11, 1, 0, 0)
```

Para obtener un objeto `datetime` de la cadena 'October 21, 2026', pasamos dicha cadena como primer argumento a `strptime()` y la cadena en formato personalizado correspondiente a esa cadena como segundo argumento 1. La cadena con información de la fecha debe coincidir a la perfección con la cadena con formato personalizado porque, de lo contrario, Python lanzará una excepción `ValueError`. Fíjate en que "November of '63" se interpreta como 2063 2 mientras que "November of '73" se interpreta como 1973 3 porque la directiva `%y` va de 1970 a 2069.

Un repaso de las funciones de tiempo de Python

Las fechas y las horas en Python pueden requerir distintos tipos de datos y funciones. Aquí tienes un repaso de los tres tipos de valores utilizados para representar el tiempo:

- Una marca de tiempo *epoch* de Unix (utilizada por el módulo `time`) es un valor flotante o entero que representa el número de segundos transcurridos desde la medianoche del 1 de enero de 1970, UTC.
- Un objeto `datetime` (del módulo `datetime`) tiene enteros almacenados en los atributos `year`, `month`, `day`, `hour`, `minute`, `second` y `microsecond`.
- Un objeto `timedelta` (del módulo `datetime`) representa una duración en vez de un momento específico.

Y aquí tienes un repaso de las funciones de tiempo, sus parámetros y sus valores de retorno:

- `time.time()`: Esta función devuelve una marca de tiempo *epoch* del momento actual como valor flotante.
- `time.sleep(seconds)`: Esta función detiene el programa durante el número de segundos especificado por el segundo argumento.
- `datetime.datetime(year, month, day, hour, minute, second)`: Esta

función devuelve un objeto `datetime` del momento especificado por los argumentos. Si no hay argumentos para hora, minuto o segundo, el valor predeterminado es 0.

- `datetime.datetime.now()`: Esta función devuelve un objeto `datetime` del momento actual.

- `datetime.datetime.fromtimestamp(epoch)`: Esta función devuelve un objeto `datetime` del momento representado por el argumento de marca de tiempo `epoch`.

- `datetime.timedelta(weeks, days, hours, minutes, seconds, milliseconds, microseconds)`: Esta función devuelve un objeto `timedelta` que representa una duración. Los argumentos de palabra clave de la función son todos opcionales y no incluyen `month` ni `year`.

- `total_seconds()`: Este método de `timedelta` devuelve el número de segundos que representa el objeto `timedelta`.

- `strftime(format)`: Este método de `datetime` devuelve una cadena en formato personalizado basándose en la cadena `format`. Véase la tabla 19.1 para los detalles de formato.

- `datetime.datetime.strptime(time_string, format)`: Esta función devuelve un objeto `datetime` que representa el momento especificado por `time_string`, analizado por el argumento de cadena `format`. Véase la tabla 19.1 para los detalles de formato.

Iniciar otros programas desde Python

Un programa de Python puede iniciar otros programas de nuestro ordenador con la función `run()` del módulo integrado `subprocess`. Si hay varias instancias de una aplicación abiertas, cada una de ellas es un proceso independiente del mismo programa. Por ejemplo, cada ventana abierta de la Calculadora en la figura 19.1 es un proceso diferente.

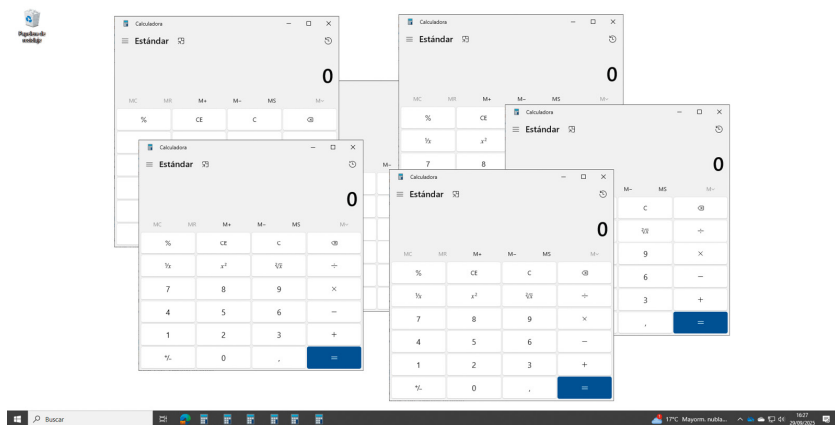


Figura 19.1. Seis procesos en ejecución del mismo programa Calculadora.

Si quieres iniciar un programa externo desde tu *script* de Python, pasa el nombre de archivo del programa a `subprocess.run()`. (En Windows, haz clic con el botón derecho en el elemento del menú Inicio y selecciona Propiedades para ver el nombre de archivo de la aplicación. En macOS, mantén pulsado **Control** mientras haces clic en la aplicación y selecciona Mostrar contenido del paquete para ver la ruta del archivo ejecutable). La función `run()` se bloqueará hasta que se cierre el programa iniciado. Pasa el programa que quieres iniciar como cadena de la ruta de archivo de su ejecutable dentro de una lista, teniendo en cuenta que el programa iniciado se ejecutará en un proceso separado, no en el mismo que tu programa de Python.

En Windows, introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > import subprocess
```

```
> > > subprocess.run(['C:\\Windows\\System32\\calc.exe'])
```

```
CompletedProcess(args=['C:\\Windows\\System32\\calc.exe'],
returncode=0)
```

En Ubuntu Linux, introduce esto:


```
> > > import subprocess
```

```
> > > subprocess.run(['usr/bin/gnome-calculator'])
```

```
CompletedProcess(args=['usr/bin/gnome-calculator'],  
returncode=0)
```

En macOS, escribe lo siguiente:

```
> > > import subprocess
```

```
> > > subprocess.run(['open', '/System/Applications/  
Calculator.app'])
```

```
CompletedProcess(args=['open', '/System/Applications/  
Calculator.app'], returncode=0)
```

Observa que macOS requiere ejecutar el programa y pasarle un argumento de línea de comandos del programa que quieres iniciar. En estos ejemplos, nuestro código Python inicia el programa, espera a que se cierre y luego continúa. Si queremos que Python inicie un programa y continúe inmediatamente, sin esperar a que se cierre ese programa, habrá que llamar a la función `subprocess.Popen()`:

```
> > > import subprocess
```

```
> > > calc_proc = subprocess.Popen(['C:\\Windows\\  
\\System32\\calc.exe'])
```

El valor de retorno es un objeto `Popen`, que tiene dos métodos útiles: `poll()` y `wait()`.

El método `poll()` es como cuando preguntas al conductor “¿llegamos ya?” una y otra vez hasta que llegas. Devolverá `None` si el proceso está todavía ejecutándose cuando se le llama. Si el programa ha terminado, `poll()` devolverá el entero del código de salida del proceso. Un código de salida indica si el proceso ha finalizado sin errores (en cuyo caso el código es 0) o si un error ha hecho que termine el proceso (representado por un número distinto de cero, por lo general 1, aunque esto puede variar en función del programa).

El método `wait()` es como esperar a que el conductor llegue a nuestro destino. El método se bloqueará hasta que el proceso iniciado termine. Esto es útil cuando se quiere que el programa se ponga en pausa hasta que el usuario termine de interactuar con el otro programa. El valor de retorno de `wait()` es el entero del código de salida del proceso.

En Windows, introduce lo siguiente en el intérprete de comandos interactivo. Ten en cuenta que la llamada a `wait()` puede bloquearse hasta que salgas de la Calculadora lanzada:

```
> > > import subprocess
```

```
1 > > > calc_proc = subprocess.Popen(['c:\\Windows\\System32\\calc.exe'])
```

```
2 > > > calc_proc.poll() = None
```

```
True
```

```
3 > > > calc_proc.wait() # Nu vuelve hasta que se cierra la Calculadora
```

```
0
```

```
> > > calc_proc.poll()
```

```
0
```

Aquí, hemos abierto un proceso del programa Calculadora 1. En versiones anteriores de Windows, `poll()` devuelve `None` 2 si el proceso sigue en ejecución. Después, cerramos la ventana de la Calculadora y, de vuelta en el intérprete de comandos interactivo, llamamos a `wait()` con el proceso terminado 3. Ahora `wait()` y `poll()` devuelven 0, indicando que el proceso ha terminado sin errores.

Si ejecutas `calc.exe` en Windows 10 o posterior usando `subprocess.Popen()`, verás que `wait()` vuelve al instante aunque la Calculadora siga en ejecución. Esto se debe a que `calc.exe` inicia la aplicación y se cierra de manera instantánea. En Windows, la Calculadora es una “aplicación de confianza de Microsoft Store” y esto queda fuera del alcance de este libro. Basta con que sepas que

hay programas que se pueden ejecutar de varias maneras específicas para la aplicación y para el sistema operativo.

Si quieres cerrar un proceso que has iniciado con `subprocess.Popen()`, llama al método `kill()` del objeto `Popen` devuelto por la función. Si tienes MS Paint en Windows, introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > import subprocess
```

```
> > > paint_proc = subprocess.Popen('c:\\Windows\\  
\\System32\\mspaint.exe')
```

```
> > > paint_proc.kill()
```

Observa que el método `kill()` termina inmediatamente un programa y omite la ventana de confirmación que pregunta si estamos seguros de que queremos salir, por lo que cualquier trabajo con el programa no guardado se perderá.

Pasar argumentos de línea de comandos a procesos

Podemos pasar argumentos de línea de comandos a los procesos que creamos con `run()`. Para ello, pasa una lista como único argumento a `run()`. La primera cadena de esta lista será el nombre de archivo del ejecutable del programa que deseas iniciar; todas las cadenas siguientes serán los argumentos de línea de comandos que pasar al programa cuando se inicie. De hecho, esta lista será el valor de `sys.argv` para el programa iniciado. La mayoría de las aplicaciones con interfaz gráfica de usuario (GUI) no utilizan argumentos de línea de comandos tanto como los programas basados en la línea de comandos o el terminal, pero la mayoría de ellas aceptarán un argumento para un archivo que abrirá la aplicación de inmediato cuando se inicie. Por ejemplo, si estás usando Windows, crea un archivo de texto `C:\\Users\\Al\\hello.txt` y escribe esto en el intérprete de comandos interactivo:

```
> > > subprocess.run(['C:\\Windows\\notepad.exe', 'C:\\  
\\Users\\Al\\hello.txt'])
```

```
CompletedProcess(args=['C:\\Windows\\notepad.exe', 'C:\\Users\\
```

```
\Al\hello.txt'], returncode = 0)
```

Esto no solo iniciará el Bloc de notas, sino que abrirá el archivo C:\Users\Al\hello.txt en el programa. Cada programa cuenta con su propio conjunto de argumentos de línea de comandos y algunos, sobre todo los que tienen GUI, no utilizan estos argumentos. La función `subprocess.Popen()` maneja los argumentos de línea de comandos de manera similar; habría que añadirlos al final de la lista pasada a la función.

Recibir texto de salida de comandos lanzados

También se puede lanzar comandos de terminal con `subprocess.run()` y `subprocess.Popen()`. Es posible que quieras que tu código Python reciba la salida de texto de estos comandos, o simule entrada de teclado para ellos. Por ejemplo, vamos a lanzar el comando `ping` desde Python y recibir el texto que produce. (Los detalles del comando `ping` quedan fuera del ámbito de este libro). En Windows, usaremos los argumentos `-n 4` para enviar cuatro solicitudes de red “ping” que comprueben si el servidor `Nostarch.com` está en línea. En macOS y Linux, sustituye `-n` por `-c`. Este comando tarda unos segundos en ejecutarse:

```
> > > import subprocess
```

```
> > > proc = subprocess.run(['ping', '-n', '4', 'nostarch.com'],  
capture_output=True, text=True)
```

```
> > > print(proc.stdout)
```

Pinging `nostarch.com` [104.20.120.46] with 32 bytes of data:

Reply from 104.20.120.46: bytes=32 time=19ms TTL=59

Reply from 104.20.120.46: bytes=32 time=17ms TTL=59

Reply from 104.20.120.46: bytes=32 time=97ms TTL=59

Reply from 104.20.120.46: bytes=32 time=217ms TTL=59

Ping statistics for 104.20.120.46:

Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),

Approximate round trip times in milli-seconds:

Minimum = 17ms, Maximum = 217ms, Average = 87ms

Cuando pasamos los argumentos `capture_output=True` y `text=True` a `subprocess.run()`, la salida de texto del programa iniciado se almacena como cadena en el atributo `stdout` del objeto

`CompletedProcess` devuelto. Nuestro *script* de Python puede usar las funciones de otros programas y analizar la salida de texto como una cadena.

Ejecutar el Programador de tareas, `launchd` y `cron`

Si te gusta la informática, es muy probable que ya conozcas el Programador de tareas de Windows, `launchd` de macOS o `cron` de Linux. Estas herramientas, bien documentadas y fiables, nos permiten programar aplicaciones para que se inicien en momentos determinados.

Utilizar el programador de tareas integrado en tu sistema operativo te evita tener que escribir tu propio código para comprobar el reloj para programar tus aplicaciones. No obstante, si solo necesitas que tu programa se ponga en pausa brevemente es mejor hacer un bucle hasta una determinada fecha y hora, llamando a `time.sleep(1)` en cada iteración por el bucle.

Abrir archivos con aplicaciones predeterminadas

Hacer doble clic en un archivo `.txt` de tu ordenador abrirá automáticamente la aplicación asociada con esa extensión de archivo. Cada sistema operativo tiene un programa que realiza el equivalente de esta acción de doble clic. En Windows, es el comando `start`. En macOS y Linux, es el comando `open`. Escribe lo siguiente en el intérprete de comandos interactivo, pasando a `run()` ‘`start`’ u ‘`open`’, dependiendo de tu sistema operativo. En Windows, deberías pasar también el argumento `shell=True`, como se muestra aquí:

```
> > > file_obj = open('hello.txt', 'w') # Crear un archivo  
hello.txt.
```

```
> > > file_obj.write('Hello, world!')
```

13

```
> > > file_obj.close()
```

```
> > > import subprocess
```

```
> > > subprocess.run(['start', 'hello.txt'], shell=True)
```

Aquí, escribimos Hello, world! en un nuevo archivo hello.txt. Después, llamamos a run(), pasándole una lista que contiene el nombre del programa o comando (en este ejemplo, 'start' para Windows) y el nombre de archivo. El sistema operativo conoce todas las asociaciones de archivos y puede suponer que debería iniciar, por ejemplo, Notepad.exe para manejar el archivo hello.txt en Windows.

Proyecto 15: Temporizador sencillo

Igual que es difícil encontrar una aplicación de cronómetro sencilla, puede costar encontrar una aplicación de temporizador sencilla. Vamos a escribir un programa que dispare una alarma al final de una cuenta atrás. Esto es lo que hará el programa:

- Una pausa de un segundo entre la muestra de cada número en el temporizador llamando a time.sleep().
- Llamar a subprocess.run() para abrir un archivo de sonido alarm.wav con la aplicación predeterminada.

Abre una nueva pestaña en el editor de archivos y guarda el programa como simplecountdown.py.

Paso 1: la cuenta atrás

Este programa requerirá el módulo time para usar la función time.sleep() y el módulo subprocess para la función

subprocess.run(). Escribe el siguiente código y guarda el archivo como simplecountdown.py:

<https://autbor.com/simplecountdown.py> - Un script para un temporizador sencillo

```
import time, subprocess
```

```
1 time_left = 60
```

```
while time_left > 0:
```

```
2 print(time_left)
```

```
3 time.sleep(1)
```

```
4 time_left = time_left - 1
```

PENDIENTE: Al final de la cuenta atrás, reproducir un archivo de sonido.

Después de importar time y subprocess, crea una variable llamada time_left para albergar el número de segundos que quedan en la cuenta atrás 1. Aquí la hemos establecido en 60, pero puedes cambiar el valor por cualquier otro o incluso establecerlo a partir de un argumento de línea de comandos. En un bucle while, muestra el tiempo que queda 2, haz una pausa de un segundo 3 y reduce la variable time_left 4 antes de que el bucle vuelva a empezar. El bucle seguirá iterando mientras time_left sea mayor que 0. Después, el temporizador acaba. A continuación, vamos a rellenar el comentario PENDIENTE con el código que reproducirá el sonido.

Paso 2: reproducir el archivo de sonido

Aunque en el capítulo 12 hemos visto cómo usar el módulo playsound3 para reproducir archivos de sonido en varios formatos, la forma más fácil y rápida de hacer esto es iniciar cualquier aplicación que ya utilice el usuario para reproducir archivos de sonido. El sistema operativo deducirá de la extensión .wav qué aplicación debería iniciar para reproducir el archivo. Este .wav podría ser cualquier otro formato de archivo de audio, como .mp3 u

.ogg. Puedes usar cualquier archivo de sonido que tengas en el ordenador al final del temporizador o descargar alarm.wav en <https://autbor.com/alarm.wav>. Añade lo siguiente a tu programa:

```
# https://autbor.com/simplecountdown.py - Un script para un temporizador sencillo
```

--fragmento omitido--

```
# Al final de la cuenta atrás, reproduce un archivo de sonido.
```

```
# subprocess.run(['start', 'alarm.wav'], shell = True) #  
Windows
```

```
# subprocess.run(['open', 'alarm.wav']) # macOS y Linux
```

Una vez que termina el bucle while, sonará alarm.wav (o el archivo de sonido que hayas elegido) para notificar al usuario que la cuenta atrás ha terminado. Quita el comentario de la llamada a subprocess.run() para tu sistema operativo. En Windows, asegúrate de incluir 'start' en la lista que pases a run(). En macOS y Linux, pasa 'open' en vez de 'start' y elimina shell = True.

En lugar de reproducir un archivo de sonido, podríamos guardar un archivo de texto en algún sitio con un mensaje como “¡Hora de descansar!” y usar subprocess.run() para abrirlo cuando termine el temporizador. Esto creará una ventana emergente con el mensaje. También podríamos usar la función webbrowser.open() para abrir una página web al final de la cuenta atrás. A diferencia de algunas aplicaciones de temporizador gratuitas que podemos encontrar en Internet, ¡la alarma de nuestro programa puede ser lo que queramos!

Ideas para programas similares

Un temporizador, básicamente, produce un simple retraso antes de continuar con la ejecución del programa. Podrías usar este enfoque para otras aplicaciones y funciones:

- Utilizar time.sleep() para dar al usuario la oportunidad de pulsar **Control-C** para cancelar una acción, como borrar archivos. El

programa puede imprimir un mensaje “Pulse Control-C para cancelar” y luego manejar las excepciones `KeyboardInterrupt` con declaraciones `try` y `except`.

- Durante un temporizador largo, puedes usar objetos `timedelta` para medir el número de días, horas minutos y segundos hasta un punto del futuro (un cumpleaños, un aniversario, etc.).

Resumen

El *epoch* de Unix (la medianoche del 1 de enero de 1970, UTC) es un tiempo de referencia estándar para muchos lenguajes de programación, incluido Python. Mientras que la función `time.time()` devuelve una marca de tiempo *epoch* (es decir, un valor flotante del número de segundos transcurridos desde el *epoch* de Unix), el módulo `datetime` es mejor para calcular fechas y formatear o analizar cadenas con información de fecha.

La función `time.sleep()` se bloqueará (es decir, no volverá) durante un número de segundos determinado. Puede servir para añadir pausas a un programa, pero, si quieres programar tus programas para que se inicien a una hora determinada, en <https://nostarch.com/automate-boring-stuff-python-3rd-edition> encontrarás instrucciones para usar el programador de tu sistema operativo.

Por último, tus programas de Python pueden iniciar otras aplicaciones con la función `subprocess.run()`. Podemos pasar argumentos de línea de comandos en la llamada a `run()` para abrir documentos específicos con la aplicación. Como alternativa, podemos usar `start` u `open` con `run()` para utilizar las asociaciones de archivo del ordenador para saber automáticamente con qué aplicación abrir un archivo. Usando otras aplicaciones del ordenador, tus programas de Python pueden potenciar sus capacidades para tus necesidades de automatización.

Preguntas prácticas

1. ¿Qué es el *epoch* de Unix?
2. ¿Qué función devuelve el número de segundos transcurridos

desde el *epoch* de Unix?

3.¿Qué función del módulo `time` devuelve una cadena legible para humanos de la hora actual, como ‘Mon Jun 15 14:00:38 2026’?

4.¿Cómo puedes pausar un programa durante exactamente cinco segundos?

5.¿Qué devuelve la función `round()`?

6.¿Cuál es la diferencia entre un objeto `datetime` y un objeto `timedelta`?

7.Usando el módulo `datetime`, ¿qué día de la semana fue el 7 de enero de 2019?

Programas de práctica

Para practicar, escribe programas que hagan las siguientes tareas.

Cronómetro con mejor aspecto

Expande el proyecto del cronómetro de este capítulo para que utilice los métodos de cadena `rjust()` y `ljust()` para que la salida tenga “mejor aspecto”. (Vimos estos métodos en el capítulo 8). En lugar de producir esto:

Lap #1: 3.56 (3.56)

Lap #2: 8.63 (5.07)

Lap #3: 17.68 (9.05)

Lap #4: 19.11 (1.43)

la salida debería ser así:

~~Lap #1:~~
Lap #1: 3.56 (3.56)

~~Lap #2:~~
Lap #2: 8.63 (5.07)

~~Lap #3:~~
Lap #3: 17.68 (9.05)

~~Lap #4:~~
Lap #4: 19.11 (1.43)

A continuación, utiliza el módulo pyperclip presentado en el capítulo 8 para copiar la salida de texto en el portapapeles para que el usuario pueda pegarla rápidamente en un archivo de texto o correo electrónico.

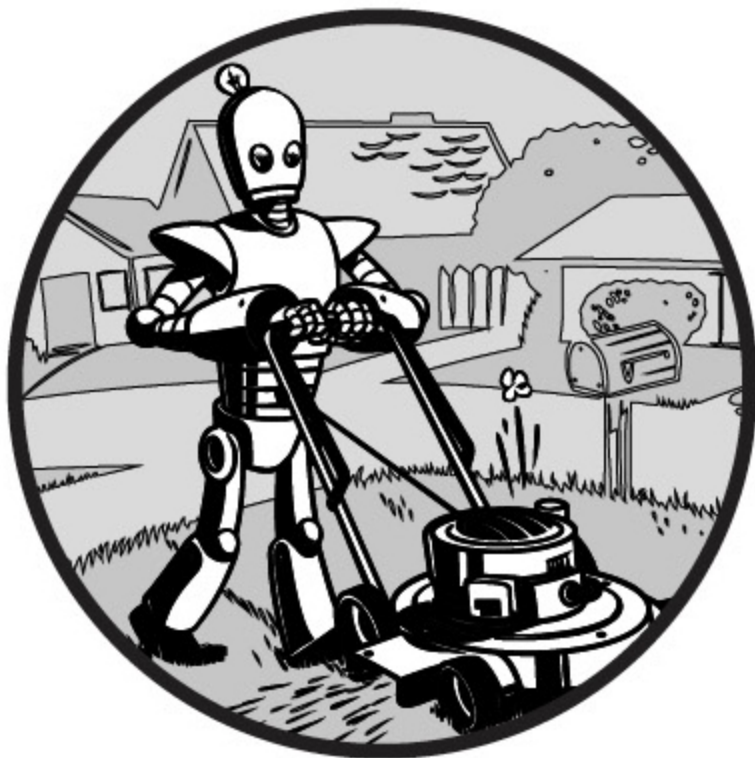
Buscador de viernes 13

En algunas culturas, el viernes 13 está considerado un día de mala suerte por los que sufren triscaidecafobia (aunque yo personalmente lo celebro como un día de buena suerte). Con los años bisiestos y meses de distinta duración, puede ser difícil saber cuándo caerá el próximo viernes 13.

Escribe dos programas. El primero debería crear un objeto datetime para la fecha actual y un objeto timedelta de un día. Si el día actual es viernes 13, debería imprimir el mes y el año. A continuación, debería sumar el objeto timedelta al objeto datetime para establecerlo en el día siguiente y repetir la comprobación. Haz que se repita hasta que haya encontrado los diez próximos viernes 13.

El segundo programa debería hacer lo mismo, pero restando el objeto timedelta. Este programa buscará todos los meses y años pasados con un viernes 13 y se detendrá cuando llegue al año 1.

Enviar correos electrónicos, textos y notificaciones push



Leer y responder correos electrónicos consume muchísimo tiempo y no podemos escribir un programa que gestione todo el correo por nosotros porque cada mensaje requiere su propia respuesta. Aun así, podrás automatizar varias tareas relacionadas con el correo electrónico cuando aprendas a escribir programas que puedan enviar y recibir correo.

Por ejemplo, tal vez tengas una hoja de cálculo llena de registros de clientes y quieras enviar a cada cliente una carta tipo diferente en función de su edad y ubicación. Es posible que el software comercial no pueda hacer esto por ti. Por suerte, puedes escribir tu

propio programa para enviar estos correos electrónicos, ahorrándote mucho tiempo de copiar y pegar.

También puedes escribir programas para enviar mensajes de texto SMS y notificaciones *push* para que te avisen de cosas incluso cuando no estés delante del ordenador. Si estás automatizando una tarea que lleva un par de horas, seguramente no querrás estar yendo al ordenador cada pocos minutos para comprobar el estado del programa. En su lugar, el programa puede enviarte un mensaje de texto al teléfono cuando haya terminado, lo que te permitirá centrarte en cosas más importantes mientras estás lejos de tu ordenador.

Este capítulo presenta el módulo EZGmail, una manera sencilla de enviar y leer correos de cuentas de Gmail, y el servicio gratuito ntfy, que proporciona notificaciones *push* entre dispositivos.

Advertencia *Te recomiendo encarecidamente crear una cuenta de correo electrónico independiente para cualquier script que envíe o reciba correos electrónicos. Así evitarás que los errores de tus programas afecten a tu cuenta de correo electrónico personal (por ejemplo, borrando mensajes o enviando spam a tus contactos sin querer). Conviene hacer primero una prueba comentando el código que realmente envía o borra correos y sustituyéndolo por una llamada temporal a print(). De ese modo, puedes probar tu programa antes de ejecutarlo de verdad.*

La API de Gmail

Gmail posee cerca de un tercio de la cuota de mercado de clientes de correo electrónico, y lo más probable es que tengas al menos una dirección suya. Debido a las medidas adicionales de seguridad y *antispam* de Gmail, es mejor controlar sus cuentas con el módulo EZGmail que con los módulos smtplib e imaplib de la biblioteca estándar de Python. Escribí EZGmail para que funcionara con la API oficial de Gmail y proporcionara funciones que facilitaran el uso de Gmail desde Python. Para obtener instrucciones completas sobre la instalación de EZGmail, consulta el apéndice A.

Habilitar la API

Antes de escribir código, debes crearte una cuenta de Gmail en <https://gmail.com>. Después, tienes que configurar la API de Gmail para tu cuenta a través de la consola de Google Cloud en <https://console.cloud.google.com>. Estos pasos son idénticos a los que seguimos para configurar EZSheets en el capítulo 15, así que no los repetiré ahora. Crea un nuevo proyecto y asegúrate de habilitar la API de Gmail en lugar de la API de Google Sheets. En el paso 2 de la configuración de la pantalla de consentimiento de OAuth, añade el ámbito <https://mail.google.com> para permitir que tus *scripts* de Python lean y envíen correos electrónicos. Cuando hayas terminado, deberías tener un archivo de credenciales y un archivo de *tokens*.

A continuación, en el intérprete de comandos interactivo, introduce el siguiente código:

```
> > > import ezgmail
```

```
> > > ezgmail.init()
```

Si no hay errores, EZGmail se ha instalado correctamente.

Enviar correos

Una vez que EZGmail esté configurado, deberías poder enviar correos con una sencilla llamada a una función:

```
> > > import ezgmail
```

```
> > > ezgmail.send('recipient@example.com', 'Subject line',  
'Body of the email')
```

Si quieres adjuntar archivos a tu mensaje, puedes proporcionar un argumento adicional a la función `send()`:

```
> > > ezgmail.send('recipient@example.com', 'Subject line',  
'Body of the email',  
['attachment1.jpg', 'attachment2.mp3'])
```

Ten en cuenta que, como parte de sus medidas de seguridad y *antispam*, Gmail podría no enviar correos repetidos con el mismo

texto exacto (porque es probable que sean *spam*) ni correos que contengan archivos .exe o .zip como adjuntos (ya que podrían ser virus). También podemos utilizar los argumentos opcionales cc y bcc para poner a otros destinatarios en copia o copia oculta:

```
> > > import ezgmail
```

```
> > > ezgmail.send('recipient@example.com', 'Subject line',  
'Body of the
```

```
email', cc='friend@example.com',  
bcc='otherfriend@example.com',
```

```
someoneelse@example.com)
```

Si necesitas recordar para qué dirección de Gmail está configurado el archivo token.json, puedes examinar ezgmail.EMAIL_ADDRESS:

```
> > > import ezgmail
```

```
> > > ezgmail.EMAIL_ADDRESS
```

```
'example@gmail.com'
```

Asegúrate de tratar el archivo token.json igual que una contraseña. Si otra persona se hace con él, puede acceder a tu cuenta de Gmail (aunque no podrán cambiar tu contraseña). Para revocar archivos token.json emitidos previamente, vuelve a la consola Google Cloud y borra la credencial del *token* comprometido. Tendrás que repetir los pasos de configuración para generar una nueva credencial y un archivo de *tokens* antes de seguir usando EZGmail.

Leer correos

Gmail organiza mensajes de correo que son respuestas a otros en hilos de conversación. Cuando inicias sesión en Gmail en tu navegador web o a través de una *app*, en realidad estás mirando hilos de correo más que mensajes individuales (incluso aunque solo haya un mensaje en el hilo). EZGmail tiene objetos GmailThread y GmailMessage para representar hilos de conversación y correos individuales, respectivamente. Un objeto GmailThread tiene un

atributo `messages` que contiene una lista de objetos `GmailMessage`. La función `unread()` devuelve una lista de objetos `GmailThread` para los 25 mensajes no leídos más recientes, que se puede pasar a `ezgmail.summary()` para imprimir un resumen de los hilos de conversación de esa lista:

```
> > > import ezgmail
```

```
> > > unread_threads = ezgmail.unread() # Lista de objetos
GmailThread
```

```
> > > ezgmail.summary(unread_threads)
```

Al, Jon - Do you want to watch RoboCop this weekend? - Dec 09

Jon - Thanks for stopping me from buying Bitcoin. - Dec 09

La función `summary()` es útil para mostrar un resumen rápido de hilos de correo, pero, para acceder a mensajes concretos (y partes de mensajes), te conviene echar un vistazo al atributo `messages` del objeto `GmailThread`. El atributo `messages` contiene una lista de los objetos `GmailMessage` que componen el hilo, y estos cuentan con atributos `subject`, `body`, `timestamp`, `sender` y `recipient` que describen el correo electrónico:

```
> > > len(unread_threads)
```

```
2
```

```
> > > str(unread_threads[0])
```

```
"<GmailThread len=2 snippet= Do you want to watch RoboCop
this weekend?>"
```

```
> > > len(unread_threads[0].messages)
```

```
2
```

```
> > > str(unread_threads[0].messages[0])
```

```
"<GmailMessage from='Al Sweigart
<al@inventwithpython.com>' to='Jon Doe
```



```
<example@gmail.com>' timestamp = datetime.datetime(2026, 12, 9, 13, 28, 48)
```

```
subject = 'RoboCop' snippet = 'Do you want to watch RoboCop this weekend?'\n>
```

```
> > > unread_threads[0].messages[0].subject
```

```
'RoboCop'
```

```
> > > unread_threads[0].messages[0].body
```

```
'Do you want to watch RoboCop this weekend?\r\n'
```

```
> > > unread_threads[0].messages[0].timestamp
```

```
datetime.datetime(2026, 12, 9, 13, 28, 48)
```

```
> > > unread_threads[0].messages[0].sender
```

```
'Al Sweigart <al@inventwithpython.com>'
```

```
> > > unread_threads[0].messages[0].recipient
```

```
'Jon Doe <example@gmail.com>'
```

Para recuperar más de los 25 correos no leídos más recientes, pasa un entero para el argumento `maxResults`. Por ejemplo, `ezgmail.unread(maxResults=50)` devolverá los 50 mensajes no leídos más recientes.

Al igual que la función `ezgmail.unread()`, la función `ezgmail.recent()` devolverá los 25 hilos más recientes de tu cuenta de Gmail:

```
> > > recent_threads = ezgmail.recent()
```

```
> > > len(recent_threads)
```

```
25
```

```
> > > recent_threads = ezgmail.recent(maxResults=100)
```

```
> > > len(recent_threads)
```

46

Puedes pasar un argumento opcional `maxResults` para cambiar este límite.

Buscar correos

Además de usar `ezgmail.unread()` y `ezgmail.recent()`, podemos buscar mensajes específicos, como haríamos si buscásemos en el cuadro de búsqueda de Gmail, llamando a `ezgmail.search()`:

```
> > > result_threads = ezgmail.search('RoboCop')
```

```
> > > len(result_threads)
```

1

```
> > > ezgmail.summary(result_threads)
```

Al, Jon - Do you want to watch RoboCop this weekend? - Dec 09

La llamada anterior a `search()` debería producir los mismos resultados que si hubiésemos escrito “RoboCop” en el cuadro de búsqueda, como en la figura 20.1.

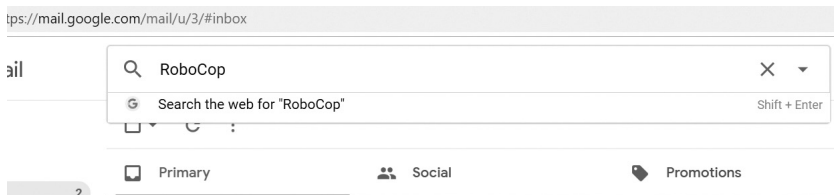


Figura 20.1. Búsqueda del correo “RoboCop” en el sitio web de Gmail.

A diferencia de `unread()` y `recent()`, la función `search()` devuelve una lista de objetos `GmailThread`. También se puede pasar a la función `search()` cualquiera de los operadores de búsqueda especiales que podemos introducir en la caja de búsqueda, así:

- ‘label:UNREAD’: Para correo no leído.
- ‘from:al@inventwithpython.com’: Para correo desde al@inventwithpython.com.
- ‘subject:hello’: Para correo con “hello” en el asunto.
- ‘has:attachment’: Para correo con archivos adjuntos.

Puedes ver una lista completa de operadores de búsqueda en <https://support.google.com/mail/answer/7190>.

Descargar adjuntos

Un objeto GmailMessage tiene un atributo attachments, que es una lista de nombres de archivo para los adjuntos al mensaje. Podemos pasar cualquiera de estos nombres al método downloadAttachment() del objeto GmailMessage. También podemos descargar todos a la vez con downloadAllAttachments(). Por defecto, EZGmail guarda los adjuntos en el directorio de trabajo actual, pero podemos pasar un argumento adicional, downloadFolder, a downloadAttachment() y downloadAllAttachments(). He aquí un ejemplo:

```
> > > import ezgmail
> > > threads = ezgmail.search('vacation photos')
> > > threads[0].messages[0].attachments
['tulips.jpg', 'canal.jpg', 'bicycles.jpg']
> > >
threads[0].messages[0].downloadAttachment('tulips.jpg')
> > >
threads[0].messages[0].downloadAllAttachments(downloadFolder=
['tulips.jpg', 'canal.jpg', 'bicycles.jpg'])
```

Si ya existe un archivo con ese nombre, el adjunto lo reemplazará automáticamente. EZGmail contiene funciones adicionales;

encontrarás toda la documentación en <https://github.com/asweigart/ezgmail>.

Pasarelas de correo a SMS

La gente suele estar más cerca de sus teléfonos que de sus ordenadores, por lo que los mensajes de texto suelen ser una forma más fiable de enviar notificaciones inmediatas que el correo electrónico. Además, los mensajes de texto suelen ser más cortos, lo que aumenta la probabilidad de que la persona los lea. La forma más fácil, aunque no la más fiable, de enviar mensajes de texto es utilizando una pasarela de correo electrónico de servicio de mensajes cortos (SMS), un servidor de correo electrónico que un proveedor de telefonía móvil ha configurado para recibir textos por correo y luego reenviarlos al destinatario como mensajes de texto.

Puedes escribir un programa para enviar estos correos electrónicos utilizando EZGmail o el módulo `smtplib`. El número de teléfono y el servidor de correo electrónico de la compañía telefónica conforman la dirección del destinatario. Por ejemplo, para enviar un mensaje de texto a un cliente de Verizon con el número de teléfono 212-555-1234, se enviaría un correo electrónico a 2125551234@vtext.com. El asunto y el cuerpo del correo electrónico aparecerían en el cuerpo del mensaje de texto.

Puedes encontrar la pasarela de correo electrónico a SMS de un proveedor de telefonía móvil buscando en Internet “pasarela correo sms *nombre del proveedor*”. La tabla 20.1 enumera las pasarelas de varios proveedores populares. Muchos de ellos tienen servidores de correo electrónico separados para SMS, que limitan los mensajes a 160 caracteres, y el servicio de mensajería multimedia (MMS), que no tiene límite de caracteres. Si deseas enviar una foto, tendrás que utilizar la pasarela a MMS y adjuntar el archivo al correo electrónico.

Si no conoces el proveedor de telefonía móvil del destinatario, puedes intentar buscarlo utilizando un sitio de búsqueda de operadores. La mejor manera de encontrar estos sitios es buscando en Internet “buscar operador de telefonía para *número*”. Muchos de estos sitios permiten buscar números de forma gratuita (aunque

cobran por buscar cientos o miles de números de teléfono a través de su API).

Tabla 20.1. Pasarelas de correo electrónico a SMS para proveedores de telefonía móvil.

Pasarelas a SMS

AT&T txtatt@txtatt.net
Boost Mobile SMS@myboostmobile.com
Cricler SMS@crickeetwickets.net
Google SMS@fi.google.com
MetroPCS SMS@metropcs.com
Public Mobile SMS@publicwireless.com
Sprint OWNS.MT@tmtpcs.com
T-Mobile SMS@tmail.net
US Cellular SMS@uscc.net
Verizon vtext@vtext.com
Virgin Mobile bl.com
Xfinity Mobile xmsmessages.com

Aunque las pasarelas de correo electrónico a SMS sean gratuitas y fáciles de usar, tienen varias desventajas importantes:

- No tenemos garantía de que el texto llegue a tiempo, ni siquiera de que llegue.
- No hay forma de saber si el texto ha llegado o no.
- El receptor no tiene manera de responder.
- Las pasarelas a SMS pueden bloquearte si envías demasiados correos y no hay forma de saber cuántos son “demasiados”.
- El que la pasarela a SMS entregue un mensaje de texto hoy no significa que vaya a funcionar mañana.

El envío de mensajes de texto a través de una pasarela SMS es ideal cuando se necesita transmitir mensajes ocasionales que no son urgentes. Si deseas una forma más fiable de enviar SMS, especialmente en grandes cantidades, puedes utilizar la API de un proveedor de servicios de telecomunicaciones como Twilio. Estos

servicios suelen requerir una suscripción o cuotas de uso, y es posible que tengas que enviar una solicitud para utilizarlos. La legislación puede variar según el país y cambiar con el tiempo.

Una alternativa al envío de SMS es utilizar un servicio gratuito de notificaciones *push*, como se explica en la siguiente sección.

Notificaciones push

Los servicios de notificación HTTP Pub/Sub permiten enviar y recibir mensajes cortos y desechables a través de Internet mediante solicitudes web HTTP. En el capítulo 13 usamos la biblioteca Requests para realizar solicitudes HTTP y aquí la utilizaremos para interactuar con el servicio en línea gratuito ntfy (que se escribe en minúsculas) en <https://ntfy.sh>. El servicio ntfy es gratuito y no requiere ningún tipo de registro.

Antes de empezar, instala la aplicación ntfy en tu teléfono móvil para poder recibir notificaciones. Esta *app* es gratuita y se encuentra en las tiendas de aplicaciones para Android y iPhone. También puedes recibir notificaciones en tu navegador web accediendo a <https://ntfy.sh/app>.

Estas aplicaciones comprueban con el servicio ntfy si hay mensajes enviados a un tema. Imagina un tema como una sala de chat o el nombre de un chat de grupo. Cualquier persona del mundo puede enviar mensajes a un tema, y cualquiera puede suscribirse a un tema para leer estos mensajes. Si quieres enviar mensajes solo para ti, utiliza un tema secreto con letras aleatorias. Trata el nombre de este tema como una contraseña y compártelo solo con aquellas personas que quieras que lean los mensajes. En este capítulo se utilizará el tema `AlSweigartZPgxBQ42` en el código de ejemplo, aunque te recomiendo que utilices tu propio tema secreto que contenga un sufijo de letras y números aleatorios. Los temas distinguen entre mayúsculas y minúsculas e incluso, si mantienes el tema en secreto, no utilices ntfy para enviar información confidencial, como contraseñas o números de tarjetas de crédito.

Para enviar una notificación *push* a todos los suscriptores de un tema, solo hay que realizar una solicitud HTTP al servidor web ntfy. Esto significa que se puede hacer íntegramente con la biblioteca Requests. No es necesario instalar ningún paquete específico para ntfy.

Para enviar la solicitud, introduce lo siguiente en el intérprete de comandos interactivo. Reemplaza el tema de ejemplo `AlSweigartZPgxBQ42` utilizado a lo largo de este capítulo por tu propio tema aleatorio secreto:

```
> > > import requests
```

```
> > > requests.post('https://ntfy.sh/AlSweigartZPgxBQ42,  
'Hello, world!')
```

```
< Response [200] >
```

Ten en cuenta que llamamos a `requests.post()` para realizar una solicitud HTTP POST para enviar una notificación. Esto es diferente de la función `requests.get()` que vimos en el capítulo 13 para descargar páginas web.

Cualquier persona suscrita al tema `AlSweigartZPgxBQ42` recibirá el mensaje “Hello, world!” en unos segundos (aunque a veces los mensajes pueden retrasarse unos minutos). También puedes verlos tú mismo en <https://ntfy.sh/AlSweigartZPgxBQ42>.

El servicio ntfy tiene algunas limitaciones. Los usuarios gratuitos tienen un límite de 250 mensajes al día, y los mensajes pueden tener un tamaño máximo de 4.096 bytes. El envío masivo de mensajes a varios temas puede hacer que tu dirección IP quede bloqueada temporalmente. Puedes obtener una cuenta de pago para aumentar estos límites en el sitio web de ntfy. Las cuentas de pago de ntfy pueden establecer temas reservados y limitar quién puede publicar en ellos. Si no tienes permiso para publicar en un tema reservado, obtendrás una respuesta `<Response [403]>` a tu llamada a la función `requests.post()`.

Dentro del límite de 4.096 bytes, tus mensajes pueden tener cualquier formato. Ten en cuenta que no hay forma de determinar

quién publicó un mensaje en un tema, por lo que tal vez te convenga incluir etiquetas “Para” y “De” en el texto de tu mensaje. O, mejor aún, podrías hacerlo utilizando JSON o algún otro formato de serialización de datos de los que vimos en el capítulo 18.

Si quieres que tus programas de Python te envíen una notificación, estas son las dos únicas líneas de código que necesitas una vez que hayas instalado la aplicación ntfy en tu teléfono y te hayas suscrito al tema. Puedes ejecutar tu programa de Python y salir a tomar un café, sabiendo que recibirás una notificación en el teléfono cuando el programa haya terminado su aburrida tarea.

Transmitir metadatos

Aunque el mensaje que envías es un valor de cadena de texto de formato libre, ntfy puede adjuntar opcionalmente valores de metadatos, como un título, un nivel de prioridad y etiquetas, a cada mensaje.

El título es similar al asunto de un correo electrónico y la mayoría de las aplicaciones lo muestran encima del texto del mensaje con una fuente más grande. El nivel de prioridad va del 1 (la prioridad más baja) al 5 (la más alta), siendo 3 el valor predeterminado. Una prioridad más alta no hace que los mensajes se entreguen más rápido, sino que permite a los suscriptores configurar sus aplicaciones de notificación para que solo muestren mensajes de una determinada prioridad o superior. Las etiquetas son palabras clave que los suscriptores pueden utilizar para filtrar los mensajes. Las etiquetas también pueden ser el nombre de un emoji que se muestra junto al título del mensaje. Puedes encontrar una lista de estos emojis en <https://docs.ntfy.sh/publish/#tags-emojis>.

Estos metadatos se incluyen en la solicitud HTTP como encabezados, por lo que tendrás que pasar un diccionario con ellos al argumento headers. Introduce lo siguiente en el intérprete de comandos interactivo para publicar un mensaje con metadatos:

```
> > > import requests
```

```
> > > requests.post('https://ntfy.sh/AlSweigartZPgxBQ42,  
'The rent is too high!')
```



```
headers = {'Title': 'Important: Read this!', 'Tags':  
'warning,neutral_face', 'Priority': '5'})
```

< Response [200] >

Estas funciones son útiles para los usuarios humanos que leen las notificaciones en la *app* de su teléfono. Sin embargo, también se puede escribir código para que los *scripts* de Python puedan recibir notificaciones *push*, como veremos en la siguiente sección.

Recibir notificaciones

Tus programas de Python también pueden leer los mensajes de un tema en particular haciendo solicitudes HTTP con la biblioteca Requests. Envía un mensaje de notificación con el código de los apartados anteriores e introduce esto en el intérprete de comandos interactivo, usando el mismo tema que en las notificaciones:

```
> > > import requests
```

```
> > > resp = requests.get('https://ntfy.sh/  
AlSweigartZPgxBQ42/json?poll=1')
```

```
> > > resp.text
```

```
{“id”:”1jnHKeFNqwnS”,”time”:1797823340,”expires”:1797866540,”event”:  
“message”,”topic”:”AlSweigartZPgxBQ42”,”message”:”Hello,  
world!”}\n
```

```
{“id”:”wZ22cJyKXw1F”,”time”:1797823712,”expires”:1797866912,”event”:  
“message”,”topic”:”AlSweigartZPgxBQ42”,”title”:”Important: Read  
this!”,
```

```
“message”:”The rent is too high!”,”priority”:5,”tags”:[“warning”,  
“neutral_face”]}\n
```

Observa que llamamos a la función `requests.get()` para recibir notificaciones, a diferencia de la función `requests.post()` que usamos para enviar notificaciones. Además, la URL termina en `json?poll=1`.

Esto es recuperar mensajes mediante sondeo, que devuelve todos los mensajes de un tema que tiene el servidor. También hay métodos de *streaming* para recuperar mensajes ntfy, pero el sondeo tiene el código más sencillo. También puedes añadir un parámetro URL `since` después de `poll = 1` para obtener los mensajes según uno de los siguientes criterios:

- `since = 10m`: Recupera todos los mensajes para el tema en los últimos 10 minutos. También podemos usar `s` para segundos y `h` para horas, como `since = 2h30m` para todos los mensajes de las dos últimas horas y media.
- `since = 1737866912`: Recupera todos los mensajes desde la marca de tiempo *epoch* de Unix de 1737866912. `time.time()` devuelve este tipo de marca temporal, que representa el número de segundos transcurridos desde el 1 de enero de 1970. El capítulo 19 trata sobre las funciones relacionadas con el tiempo.
- `since = wZ22cjyKXw1F`: Recupera todos los mensajes después del que tenía el ID `'wZ22cjyKXw1F'`.

Separa los parámetros URL adicionales con un símbolo `&`. Por ejemplo, al pasar la URL <https://ntfy.sh/AlSweigartZPgxBQ42/json?poll=1&since=10m> se recuperan todos los mensajes del tema `AlSweigartZPgxBQ42` de los últimos 10 minutos. Para reducir la carga en el servidor ntfy, deberías realizar sondeos solo una vez por minuto o cada pocos minutos, en lugar de hacerlo lo más rápido posible en un bucle infinito. Si necesitas recibir notificaciones de inmediato, consulta la documentación en línea para obtener información sobre cómo suscribirse a flujos de notificaciones.

El texto de esta respuesta HTTP no es JSON válido, ya que contiene un objeto JSON en cada línea de texto, en lugar de un solo objeto JSON, por lo que utilizamos el método de cadena `splitlines()` antes de analizarlos individualmente con el módulo `json` (como se explica en el capítulo 18). Continúa con el ejemplo anterior en el intérprete de comandos interactivo:

```
> > > import json
```

```
> > > notifications = ]
```

```
> > > for json_text in resp.text.splitlines():  
... notifications.append(json.loads(json_text))
```

.

```
> > > notifications[0] ['message']
```

'Hello, world!'

```
> > > notifications[1] ['message']
```

'The rent is too high!'

La función `json.loads()` convierte el texto JSON de ntfy en un diccionario de Python. Echemos un vistazo a cada par clave-valor:

- “id”:“wZ22cjyKXw1F”: El valor de la clave ‘id’ es una cadena de identificación única que ayuda a diferenciar varias notificaciones incluso aunque tengan el mismo texto.

- “time”:1797823712: El valor de la clave ‘time’ es una marca de tiempo *epoch* de Unix del momento en el que se creó la notificación. Llamar a `str(datetime.datetime.fromtimestamp(1797823712))` devuelve la cadena legible para humanos ‘2026-12-20 21:28:32’.

- “expires”:1797866912: El valor de la clave ‘expires’ es una marca de tiempo *epoch* de Unix del momento en el que la notificación se eliminará del servidor ntfy.

- “event”:“message”: El valor de la clave ‘event’ puede ser ‘message’, ‘open’, ‘keepalive’ o ‘poll_request’. Estos tipos de evento se explican en la documentación en línea, pero, por ahora, es probable que el único que te interese sea ‘message’.

- “topic”:“AlSweigartZPgxBQ42”: La parte correspondiente al tema en la URL se repite en el valor de la clave ‘topic’.

- “title”:“Important: Read this!”: Si la notificación tiene título, habrá una clave ‘title’ que lo contenga como valor de cadena.

- “message”:“The rent is too high!”: El valor de la clave ‘message’ es

una cadena del texto de la notificación.

- “priority”:5: Si la notificación tiene una prioridad, habrá una clave ‘priority’ con un valor entero de 1 a 5.

- “tags”:[“warning”, “neutral_face”]: Si la notificación tiene etiquetas, habrá una clave ‘tags’ con ellas como lista de valores. Estos valores de cadena pueden ser los nombres de los emojis que mostrar.

Al leer los valores de este diccionario, tus programas de Python pueden utilizar la biblioteca Requests para recibir notificaciones realizadas por usuarios u otros *scripts* de Python. El servicio ntfy es una de las formas más sencillas de crear programas que puedan comunicarse entre sí a través de Internet (aunque hay que tener en cuenta el límite de 250 mensajes al día para los usuarios gratuitos).

Resumen

Nos comunicamos entre nosotros a través de Internet y redes de telefonía móvil de decenas de formas diferentes, pero predominan el correo electrónico y los mensajes de texto. Tus programas pueden comunicarse a través de estos canales, lo que les proporciona nuevas y potentes funciones de notificación.

Como medida de seguridad y *antispam*, algunos servicios de correo electrónico populares, como Gmail, no permiten utilizar los protocolos SMTP e IMAP estándar para acceder a sus servicios. El paquete EZGmail actúa como un práctico envoltorio para la API de Gmail, lo que permite que nuestros *scripts* de Python accedan a nuestra cuenta de Gmail. Te recomiendo encarecidamente configurar una cuenta independiente para que la utilicen tus *scripts*, de modo que los posibles errores de un programa no den problemas en tu cuenta personal de Gmail.

Los mensajes de texto son un poco diferentes del correo electrónico, ya que, a diferencia de este, necesitarás algo más que una conexión a Internet para enviar SMS. Puedes utilizar pasarelas de correo electrónico a SMS para enviar mensajes de texto desde una cuenta de correo electrónico, aunque esto requiere conocer el operador de telecomunicaciones del usuario del teléfono y no es una forma fiable de enviar mensajes. Si solo te envías mensajes cortos a ti

mismo, puedes utilizar el sistema de notificaciones push en <https://ntfy.sh> y, después, instalar la aplicación ntfy en tu teléfono para que tus *scripts* de Python envíen mensajes a los suscriptores del tema.

Con estos módulos en tu caja de herramientas, podrás programar las condiciones específicas en las que tus programas deben enviar notificaciones o recordatorios. ¡Ahora tus programas tendrán un alcance que va mucho más allá del ordenador en el que se ejecutan!

Preguntas prácticas

1. Cuando se usa la API de Gmail, ¿qué son los archivos de credenciales y *tokens*?
2. En la API de Gmail, ¿cuál es la diferencia entre objetos “hilo” y objetos “mensaje”?
3. ¿Cómo podemos buscar correos electrónicos con adjuntos utilizando `ezgmail.search()`?
4. ¿Cuáles son algunas de las desventajas de usar pasarelas de correo a SMS para enviar mensajes de texto?
5. ¿Qué biblioteca de Python puede enviar notificaciones a ntfy y recibirlas?

Programas de práctica

Para practicar, escribe programas que realicen las siguientes tareas.

Recordatorio de paraguas

En el capítulo 13 vimos cómo usar el módulo `requests` para obtener datos de <https://weather.gov>. Escribe un programa que se ejecute antes de que te levantes por la mañana y compruebe la probabilidad de lluvia en la predicción meteorológica para ese día. Si va a llover, haz que el programa te envíe un recordatorio para coger un paraguas antes de salir de casa.

Cancelador de suscripciones automático

Escribe un programa que mire tu correo, busque todos los enlaces para cancelar suscripciones en tu mensaje y los abra automáticamente en un navegador. Este programa tendrá que entrar en tu cuenta de Gmail. Puedes usar BeautifulSoup (visto en el capítulo 13) para buscar todas las apariciones de la expresión “cancelar suscripción” dentro de una etiqueta de enlace de HTML. Cuando tengas una lista de estas URL, puedes usar `webbrowser.open()` para abrir automáticamente todos estos enlaces en un navegador.

Tendrás que realizar los siguientes pasos a mano para cancelar las suscripciones de estas listas. En la mayoría de los casos, eso implica hacer clic en un enlace para confirmar, pero este *script* te evita tener que repasar todos tus correos en busca de enlaces para cancelar suscripciones.

Control informático basado en correo electrónico

Escribe un programa que compruebe cada 15 minutos tu cuenta de correo electrónico o ntfy en busca de instrucciones que le envíes y las ejecute automáticamente. Por ejemplo, BitTorrent es un sistema de descarga entre pares. Con un software gratuito de BitTorrent, como qBittorrent, puedes descargar archivos multimedia de gran tamaño en tu ordenador personal. Si envías al programa un enlace BitTorrent (totalmente legal, nada pirata), el programa comprobará su correo electrónico o buscará notificaciones ntfy, encontrará este mensaje, extraerá el enlace y, a continuación, iniciará qBittorrent para comenzar a descargar el archivo. De esta manera, puedes hacer que tu ordenador personal comience las descargas mientras estás fuera y terminar la descarga (completamente legal, nada pirata) para cuando regreses a casa.

El capítulo 19 explica cómo iniciar programas en el ordenador con la función `subprocess.Popen()`. Por ejemplo, la siguiente llamada iniciaría el programa qBittorrent, con un archivo torrent:

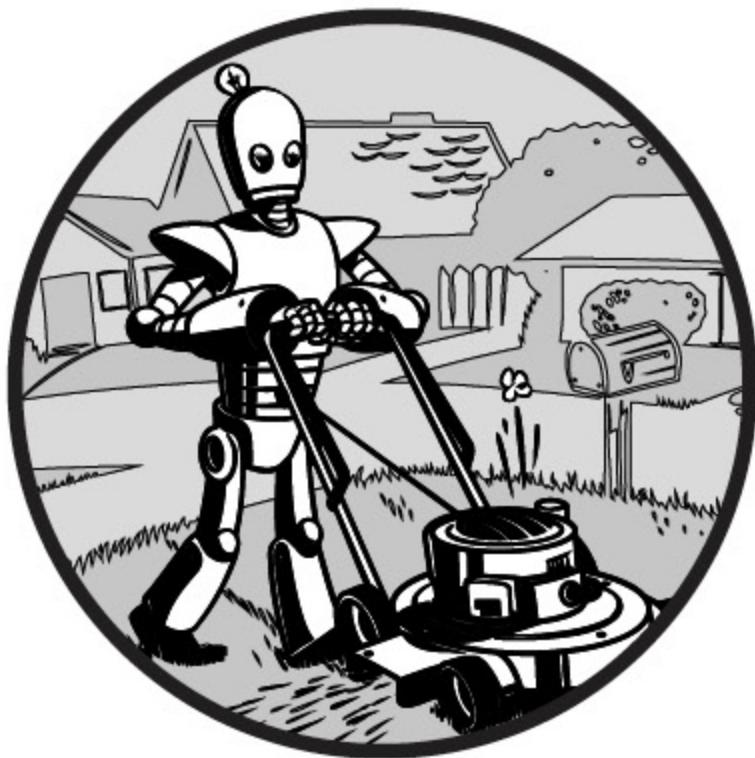
```
qbProcess = subprocess.Popen(['C:\\Program Files (x86)\\  
qBittorrent\\  
qbittorrent.exe', 'shakespeare_complete_works.torrent'])
```

Naturalmente, te conviene que el programa se asegure de que los correos electrónicos provienen de ti. En particular, es posible que desees exigir que los correos electrónicos contengan una contraseña, ya que para los piratas informáticos es bastante fácil falsificar la dirección del remitente en los correos electrónicos. El programa debe eliminar los correos que encuentre para no repetir las instrucciones cada vez que compruebe la cuenta. Como función adicional, haz que el programa te envíe un correo electrónico o un mensaje de texto de confirmación cada vez que ejecute un comando. Dado que no estarás sentado frente al ordenador que ejecuta el programa, es una buena idea utilizar las funciones de registro (repasa el capítulo 5) para escribir un archivo de texto de registro que puedas consultar si se producen errores.

El programa qBittorrent (al igual que otras aplicaciones BitTorrent) tiene una función que le permite cerrarse automáticamente una vez completada la descarga. En el capítulo 19 vimos cómo se puede determinar cuándo se ha cerrado una aplicación iniciada con el método `wait()` para objetos `Popen`. La llamada al método `wait()` se bloqueará hasta que qBittorrent se haya detenido, y entonces tu programa podrá enviarte un correo electrónico o un mensaje de texto con una notificación de que la descarga se ha completado.

Hay muchas funciones que se pueden añadir a este proyecto. Si te quedas atascado, puedes descargar un ejemplo de implementación del programa en la página web del libro.

Hacer gráficos y manipular imágenes



Si tienes una cámara digital o subes fotos desde tu teléfono a una red social, probablemente te encuentres con archivos de imágenes digitales constantemente. Quizás sepas cómo usar programas básicos de edición de imágenes, como Microsoft Paint o Paintbrush, o incluso aplicaciones más avanzadas, como Adobe Photoshop. Pero, si necesitas editar una gran cantidad de imágenes, modificarlas a mano puede ser una tarea larga y aburrida.

Vas a conocer Pillow, un paquete Python de terceros para interactuar con archivos de imagen. Este paquete tiene varias funciones que hacen más fácil recortar, cambiar el tamaño y editar

el contenido de una imagen. En este capítulo se explica cómo utilizar Pillow para que Python pueda editar automáticamente cientos o miles de imágenes con facilidad.

Este capítulo también habla sobre Matplotlib, una popular biblioteca para crear gráficos de aspecto profesional. Matplotlib cuenta con muchas funciones y opciones personalizables, y hay numerosos libros dedicados íntegramente a ella. Aquí trataremos los conceptos básicos de la generación de imágenes gráficas con Matplotlib.

Fundamentos de la imagen digital

Para manipular una imagen, debes comprender cómo trabajar con colores y coordenadas en Pillow. Puedes instalar la última versión de Pillow siguiendo las instrucciones del apéndice A.

Colores y valores RGBA

Los programas de ordenador suelen representar un color en una imagen como un valor RGBA, un grupo de números que especifican la cantidad de rojo, verde, azul y alfa (o transparencia) que incluye. Cada uno de estos valores es un número entero que va de 0 (ninguno) a 255 (el máximo). Estos valores RGBA pertenecen a píxeles individuales, el punto más pequeño de un solo color que puede mostrar la pantalla del ordenador. La configuración RGB de un píxel le indica con precisión qué tono de color debe mostrar. Si una imagen en la pantalla se superpone a una imagen de fondo o al fondo de escritorio, el valor alfa determina cuánto del fondo se puede “ver a través” del píxel de la imagen. Pillow representa los valores RGBA con una tupla de cuatro valores enteros. Por ejemplo, representa el color rojo con (255, 0, 0, 255). Este color tiene la cantidad máxima de rojo, sin verde ni azul, y el valor alfa máximo, lo que significa que es totalmente opaco. Pillow representa el verde con (0, 255, 0, 255) y el azul con (0, 0, 255, 255). El blanco, la combinación de todos los colores, es (255, 255, 255, 255), mientras que el negro, que no tiene ningún color, es (0, 0, 0, 255).

Si un color tiene un valor alfa de 0, da igual cuáles sean los valores R. Después de todo, el rojo invisible es igual que el negro invisible.

Pillow utiliza los mismos nombres de colores que HTML. La tabla 21.1 recoge una selección de nombres de colores estándar y sus valores.

Tabla 21.1. Nombres de colores estándar y sus valores RGBA.

Nombre RGBA

White (255, 255, 255)

Black (0, 0, 0)

Blue (0, 0, 255)

Red (255, 0, 0)

Green (0, 255, 0)

Pillow ofrece la función `ImageColor.getcolor()` para que no tengamos que memorizar los valores RGBA de los colores que queremos usar. Esta función toma una cadena del nombre del color como primer argumento y la cadena 'RGBA' como segundo argumento para devolver una tupla RGBA. Para ver cómo funciona, introduce lo siguiente en el intérprete de comandos interactivo:

```
1 >>> from PIL import ImageColor
```

```
2 >>> ImageColor.getcolor('red', 'RGBA')
```

```
(255, 0, 0, 255)
```

```
3 >>> ImageColor.getcolor('RED', 'RGBA')
```

```
(255, 0, 0, 255)
```

```
>>> ImageColor.getcolor('Black', 'RGBA')
```

```
(0, 0, 0, 255)
```

```
>>> ImageColor.getcolor('chocolate', 'RGBA')
```

```
(210, 105, 30, 255)
```

```
>>> ImageColor.getcolor('CornflowerBlue', 'RGBA')
```

```
(100, 149, 237, 255)
```

Primero, importamos el módulo `ImageColor` desde `PIL` (no `Pillow`,

por una historia de nomenclatura que queda fuera del alcance de este libro) 1. La cadena del nombre de color que pasamos a `ImageColor.getcolor()` no distingue mayúsculas y minúsculas, por lo que `'red'` 2 y `'RED'` 3 nos dan la misma tupla RGBA. También podemos pasar nombres de colores más inusuales como `'chocolate'` o `'CornflowerBlue'`.

Pillow soporta una lista enorme de nombres de colores, desde `'aliceblue'` hasta `'yellowgreen'`. Escribe esto en el intérprete de comandos interactivo para ver los nombres de colores:

```
> > > from PIL import ImageColor
```

```
> > > list(ImageColor.colormap)
```

```
['aliceblue', 'antiquewhite', 'aqua', ... 'yellow', 'yellowgreen']
```

Encontrarás la lista completa con más de 100 nombres de colores estándar en las claves del diccionario `ImageColor.colormap`.

Coordenadas y tuplas de cuadros

Los píxeles de imágenes se indican con coordenadas x e y , que especifican la posición horizontal y vertical, respectivamente, en una imagen. El origen es el píxel de la esquina superior izquierda de la imagen, especificado con la notación $(0, 0)$. El primer cero representa la coordenada x , que empieza en cero en el origen y aumenta de izquierda a derecha. El segundo cero representa la coordenada y , que empieza en cero en el origen y aumenta hacia abajo. Conviene repetirlo: las coordenadas y aumentan hacia abajo, que es justo lo contrario de lo que seguramente recuerdas de las coordenadas en las clases de matemáticas. La figura 21.1 muestra cómo funciona este sistema.

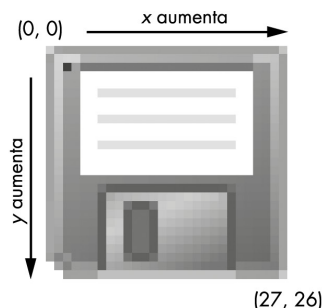


Figura 21.1. Las coordenadas x e y de una imagen de 28×27 de un dispositivo de almacenamiento de datos antiguo.

Muchos de los métodos y funciones de Pillow toman un argumento de tupla de cuadro. Esto significa que Pillow espera una tupla de cuatro coordenadas enteras que representan una región rectangular en una imagen. Los cuatro enteros son, por orden, estos:

- Izquierdo:** La coordenada x del extremo izquierdo del cuadro.
- Superior:** La coordenada y del extremo superior del cuadro.
- Derecho:** La coordenada x de un píxel a la derecha del extremo derecho del cuadro. Este entero debe ser mayor que el izquierdo.
- Inferior:** La coordenada y de un píxel más abajo que el extremo inferior del cuadro. Este entero debe ser mayor que el superior.

Ten en cuenta que el cuadro incluye las coordenadas izquierda y superior, pero no la derecha y la inferior. Por ejemplo, la tupla $(3, 1, 9, 6)$ representa todos los píxeles del recuadro negro de la figura 21.2.

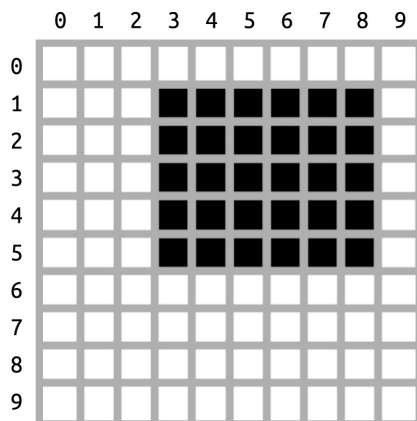


Figura 21.2. El área representada por la tupla (3, 1, 9, 6).

Ahora que ya sabes cómo funcionan los colores y las coordenadas en Pillow, vamos a usarlo para manipular una imagen.

Manipular imágenes con Pillow

Para practicar con Pillow, utilizaremos el archivo de imagen `zophie.png` que muestra la figura 21.3. Puedes descargarlo de los recursos en línea del libro. Guarda el archivo en tu directorio de trabajo actual y carga la imagen en Python, así:

```
> > > from PIL import Image  
  
> > > cat_im = Image.open('zophie.png')  
  
> > > cat_im.show()
```

Importa el módulo `Image` de Pillow y llama a `Image.open()`, pasándole el nombre del archivo. Después, puedes guardar la imagen cargada en una variable, como `cat_im`. Los objetos `Image` de Pillow cuentan con un método `show()` que abre la imagen en una ventana, lo cual es útil cuando se está depurando un programa y hay que identificar la imagen de un objeto `Image`.



Figura 21.3. Mi gata, Zophie.

Si el archivo de imagen no está en el directorio de trabajo actual, cambia este a la carpeta que contenga la imagen llamando a la función `os.chdir()`:

```
> > > import os
```

```
> > > os.chdir('C:\\folder_with_image_file')
```

La función `Image.open()` devuelve un valor del tipo de datos `Image` que usa Pillow para representar una imagen como valor de Python. Podemos cargar un objeto `Image` a partir de una imagen en cualquier formato pasando a la función `Image.open()` una cadena del nombre del archivo. Cualquier cambio realizado en el objeto `Image` se puede guardar en un archivo de imagen (también en cualquier formato) con el método `save()`. Todas las operaciones de rotación, cambio de tamaño, recorte, dibujo y otras manipulaciones se hacen con llamadas a métodos con este objeto `Image`.

Para acortar los ejemplos de este capítulo, asumiré que has importado el módulo `Image` de Pillow y guardado la imagen de Zophie en una variable llamada `cat_im`. Asegúrate de que el archivo `zophie.png` está en el directorio de trabajo actual para que la

función `Image.open()` lo encuentre. De lo contrario, tendrás que especificar la ruta absoluta en el argumento de cadena de la función.

Trabajar con el tipo de datos `Image`

Un objeto `Image` tiene varios atributos útiles que nos dan información básica del archivo de imagen desde el que se ha cargado: su anchura y altura, el nombre del archivo y el formato (como JPEG, WebP, GIF o PNG). Para ver un ejemplo, introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > from PIL import Image
> > > cat_im = Image.open('zophie.png')
> > > cat_im.size
1 (816, 1088)
2 > > > width, height = cat_im.size
3 > > > width
816
4 > > > height
1088
> > > cat_im.filename
'zophie.png'
> > > cat_im.format
'PNG'
> > > cat_im.format_description
'Portable network graphics'
```

```
5 > > > cat_im.save('zophie.jpg')
```

Después de crear un objeto Image a partir de zophie.png y guardarlo en cat_im, el atributo size del objeto contiene una tupla de la anchura y la altura de la página en píxeles 1. Podemos asignar los valores de la tupla a variables width y height 2 para acceder a la anchura 3 y la altura 4 de manera individual. El atributo filename describe el nombre del archivo original. Los atributos format y format_description son cadenas que describen el formato de imagen del archivo original (siendo format_description un poco más verboso).

Por último, llamar al método save() y pasarle 'zophie.jpg' guarda una nueva imagen con el nombre zophie.jpg en el disco duro 5. Pillow ve que la extensión de archivo es .jpg y guarda automáticamente la imagen con el formato JPEG. Ahora deberíamos tener dos imágenes en el disco duro: zophie.png y zophie.jpg. Aunque estos archivos se basan en la misma imagen, no son idénticos porque tienen distinto formato.

Pillow también cuenta con la función Image.new(), que devuelve un objeto Image, parecido a Image.open(), con la salvedad de que la imagen representada por el objeto Image.new() estará en blanco. Estos son los argumentos para Image.new():

- La cadena 'RGBA', que establece el modo de color en RGBA. (Hay otros modos en los que no vamos a entrar en este libro).
- El tamaño como una tupla de dos enteros para la anchura y la altura de la imagen.
- El color de fondo con el que debería empezar la imagen como una tupla de cuatro enteros de un valor RGBA. Puedes usar el valor de retorno de la función ImageColor.getcolor() para este argumento. Como alternativa, puedes pasar el nombre del color estándar como cadena.

Por ejemplo, introduce esto en el intérprete de comandos interactivo:

```
> > > from PIL import Image
```



```
1 >>> im = Image.new('RGBA', (100, 200), 'purple')
```

```
>>> im.save('purpleImage.png')
```

```
2 >>> im2 = Image.new('RGBA', (20, 20))
```

```
>>> im2.save('transparentImage.png')
```

Aquí, podemos crear un objeto Image para una imagen que tiene 100 píxeles de ancho y 200 píxeles de alto, con un fondo morado 1. Después, guardamos la imagen en el archivo purpleImage.png.

Volvemos a llamar a Image.new() para crear otro objeto Image, esta vez pasando (20, 20) para las dimensiones y nada para el color de fondo 2. Negro invisible, (0, 0, 0, 0), es el color predeterminado si no se especifica ningún argumento de color, así que la segunda imagen tiene un fondo transparente. Guardamos este cuadrado transparente de 20×20 en transparentImage.png.

Recortar imágenes

Recortar una imagen significa seleccionar una región rectangular dentro de una imagen y quitar todo lo que queda fuera del rectángulo. El método crop() de los objetos Image coge una tupla de cuadro y devuelve un objeto Image que representa la imagen recortada.

El recorte no se hace *in situ*, es decir, el objeto Image original permanece intacto, lo que devuelve el método crop() es uno nuevo. Recuerda que una tupla de cuadro (en este caso, la sección de recorte) incluye la columna izquierda y la fila superior de píxeles, pero no la columna derecha ni la fila inferior.

Escribe esto en el intérprete de comandos interactivo:

```
>>> from PIL import Image
```

```
>>> cat_im = Image.open('zophie.png')
```

```
>>> cropped_im = cat_im.crop((335, 345, 565, 560))
```

```
>>> cropped_im.save('cropped.png')
```

Este código crea un nuevo objeto Image para la imagen recortada, lo guarda en `cropped_im` y luego llama a `save()` con `cropped_im` para guardar la imagen recortada en `cropped.png`, que se muestra en la figura 21.4.

El recorte crea el nuevo archivo a partir del original.

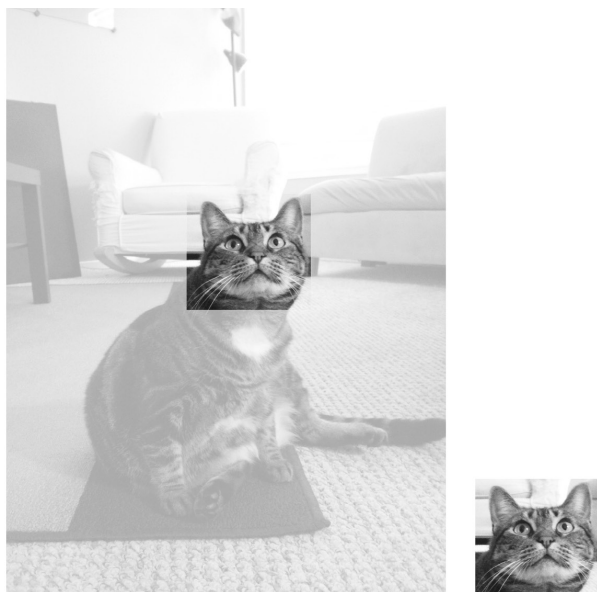


Figura 21.4. La nueva imagen es la sección recortada de la original.

Pegar imágenes en otras imágenes

El método `copy()` devolverá un nuevo objeto Image que contiene la misma imagen que el objeto Image con el que se ha llamado. Esto es útil si necesitas hacer cambios en una imagen, pero quieres conservar una versión intacta del original. Por ejemplo, introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > from PIL import Image  
> > > cat_im = Image.open('zophie.png')  
> > > cat_copy_im = cat_im.copy()
```

Las variables `cat_im` y `cat_copy_im` contienen dos objetos `Image` separados que contienen la misma imagen. Ahora que tenemos un objeto `Image` almacenado en `cat_copy_im`, podemos modificar `cat_copy_im` como queramos y guardarlo con otro nombre de archivo, dejando `cat_im` como estaba.

Cuando se llama con un objeto `Image`, el método `paste()` pega otra imagen encima. Sigamos con el ejemplo del intérprete de comandos interactivo pegando una imagen más pequeña en `cat_copy_im`:

```
> > > face_im = cat_im.crop((335, 345, 565, 560))
```

```
> > > face_im.size
```

```
(230, 215)
```

```
> > > cat_copy_im.paste(face_im, (0, 0))
```

```
> > > cat_copy_im.paste(face_im, (400, 500))
```

```
> > > cat_copy_im.save('pasted.png')
```

Primero, pasamos a `crop()` una tupla de cuadro para la zona rectangular de `zophie.png` que contiene la cara de Zophie. La llamada a este método crea un objeto `Image` que representa un recorte de 230×215 y que guardamos en `face_im`. Ahora podemos pegar `face_im` en `cat_copy_im`. El método `paste()` toma dos argumentos: un objeto `Image` de origen y una tupla de las coordenadas x e y donde queremos pegar la esquina superior izquierda de este objeto en el objeto `Image` principal. Aquí, llamamos a `paste()` dos veces con `cat_copy_im`, pegando dos copias de `face_im` en `cat_copy_im`. Por último, guardamos la versión modificada de `cat_copy_im` en `pasted.png`, que se muestra en la figura 21.5.

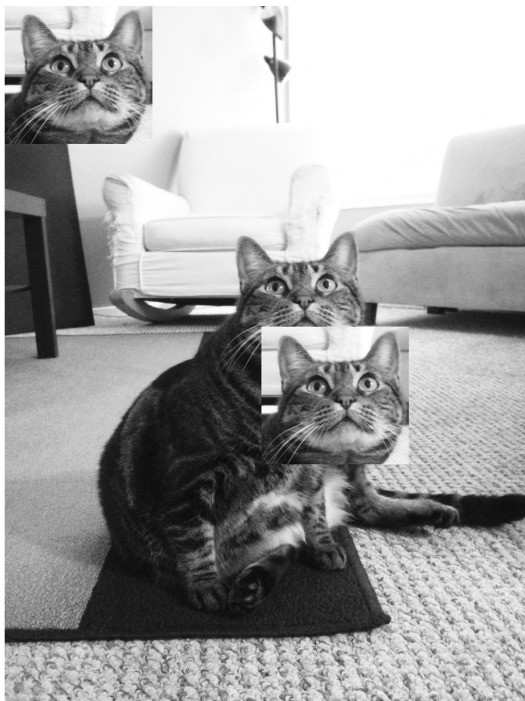


Figura 21.5. Zophie la gata, con su cara pegada dos veces.

Nota Pese a sus nombres, los métodos `copy()` y `paste()` de *Pillow* no usan el portapapeles del ordenador.

El método `paste()` modifica su objeto *Image* *in situ*; no devuelve un objeto *Image* con la imagen pegada. Si quieres llamar a `paste()` pero también conservar una versión intacta de la imagen original, primero tendrás que copiar la imagen y luego llamar a `paste()` con esa copia.

Supongamos que queremos hacer un mosaico con la cabeza de Zophie por toda la imagen, como en la figura 21.6.



Figura 21.6. Usar bucles for anidados con paste() puede duplicar la cara de la gata.

Puedes conseguir este efecto con un par de bucles for. Continúa con el ejemplo del intérprete de comandos interactivo escribiendo lo siguiente:

```
> > > cat_im_width, cat_im_height = cat_im.size  
> > > face_im_width, face_im_height = face_im.size  
1 > > > cat_copy_im = cat_im.copy()  
2 > > > for left in range(0, cat_im_width, face_im_width):  
. 3 for top in range(0, cat_im_height, face_im_height):
```

```
. print(left, top)

. cat_copy_im.paste(face_im, (left, top))
```

```
.
0 0
0 215
0 430
0 645
0 860
0 1075
230 0
230 215
```

--fragmento omitido--

```
690 860
690 1075
```

```
> > > cat_copy_im.save('tiled.png')
```

Almacenamos la anchura y la altura de la imagen original en `cat_im_width` y `cat_im_height`. A continuación, hacemos una copia de la imagen y la guardamos en `cat_copy_im` 1. Ahora podemos pasar en bucle para pegar `face_im` en la copia. La variable `left` del bucle `for` externo empieza en 0 y aumenta en `face_im_width` 2. La variable `top` del bucle `for` interno empieza en 0 y aumenta en `face_im_height` 3. Estos bucles `for` anidados producen valores para `left` y `top` que pegan una cuadrícula de imágenes `face_im` por todo el objeto `Image`, como en la figura 21.6. Para ver los bucles `for` anidados en funcionamiento, imprimimos `left` y `top`. Cuando se termina de pegar todo, guardamos la modificación de `cat_copy_im` en `tiled.png`.

Si pegamos una imagen con transparencia, tendremos que pasar la imagen como tercer argumento opcional para indicar a Pillow qué partes de la imagen original pegar. Si no lo hacemos, los píxeles transparentes de la imagen original aparecerán como píxeles blancos en la imagen pegada. Veremos esta práctica con más detalle en el proyecto 16.

Cambiar el tamaño de imágenes

Cuando se llama con un objeto Image, el método `resize()` devuelve un nuevo objeto Image con la altura y anchura especificadas. Acepta un argumento de tupla de dos enteros que representa las nuevas dimensiones. Introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > from PIL import Image

> > > cat_im = Image.open('zophie.png')

1 > > > width, height = cat_im.size

2 > > > quarter_sized_im = cat_im.resize((int(width / 2),
int(height / 2)))

> > > quarter_sized_im.save('quartersized.png')

3 > > > svelte_im = cat_im.resize((width, height + 300))

> > > svelte_im.save('svelte.png')
```

Asignamos los dos valores de la tupla `cat_im.size` a las variables `width` y `height`. Usar estas variables en vez de `cat_im.size[0]` y `cat_im.size[1]` hace el resto del código más legible. La primera llamada a `resize()` pasa `int(width / 2)` para la nueva anchura e `int(height / 2)` para la nueva altura, de modo que el objeto Image devuelto por `resize()` será la mitad de alto y de ancho que la imagen original, o un cuarto del tamaño total de la imagen original. El método `resize()` solo acepta enteros en su argumento de tupla; por eso, hay que incluir las divisiones entre 2 en una llamada a `int()`.

El cambio de tamaño mantiene las proporciones de la imagen

original, pero los nuevos valores de altura y anchura no tienen por qué conservarlas. La variable `svelte_im` contiene un objeto `Image` que tiene la anchura original, pero una altura de 300 píxeles más 3, lo que da a Zophie un aspecto más esbelto.

Observa que el método `resize()` no edita el objeto `Image`, sino que devuelve otro objeto `Image`.

Rotar y voltear imágenes

Para girar imágenes, utilizamos el método `rotate()`, que devuelve un nuevo objeto `Image` y deja el original intacto. El método acepta un solo entero o flotante que representa los grados que hay que rotar la imagen en sentido antihorario. Escribe esto en el intérprete de comandos interactivo:

```
> > > from PIL import Image
> > > cat_im = Image.open('zophie.png')
> > > cat_im.rotate(90).save('rotated90.png')
> > > cat_im.rotate(180).save('rotated180.png')
> > > cat_im.rotate(270).save('rotated270.png')
```

Observa que puedes encadenar llamadas a métodos llamando a `save()` directamente con el objeto `Image` devuelto por `rotate()`. La primera cadena `rotate()` y `save()` rota la imagen 90 grados en sentido antihorario y la guarda como `rotated90.png`. La segunda y la tercera llamadas hacen lo mismo, salvo que rotan la imagen 180 y 270 grados, respectivamente. La figura 21.7 muestra el resultado.



Figura 21.7. La imagen original (izquierda) y la imagen girada en

sentido antihorario 90, 180 y 270 grados.

Las imágenes rotadas tendrán la misma altura y anchura que la original. En Windows, un fondo negro rellenará cualquier hueco que deje la rotación, como en la figura 21.8. En macOS y Linux, el relleno se hará con píxeles transparentes.

El método `rotate()` cuenta con un argumento `expand` opcional que se puede establecer en `True` para aumentar las dimensiones de la imagen y que quepa toda la imagen nueva. Por ejemplo, introduce esto en el intérprete de comandos interactivo:

```
> > > cat_im.rotate(6).save('rotated6.png')
```

```
> > > cat_im.rotate(6,  
expand=True).save('rotated6_expanded.png')
```

La primera llamada gira la imagen seis grados y la guarda en `rotated6.png`. (Observa la imagen de la izquierda en la figura 21.8). La segunda llamada rota la imagen seis grados, establece `expand` en `True` y guarda la imagen en `rotated6_expanded.png`. (Mira la imagen de la derecha en la figura 21.8).



Figura 21.8. La imagen rotada seis grados sin más (izquierda) y con `expand=True` (derecha).

Si giramos la imagen 90, 180 o 270 grados con `expand = True`, la imagen rotada no tendrá fondo negro ni transparente.

También podemos voltear horizontalmente una imagen, como en la figura 21.9, con el método `transpose()`.

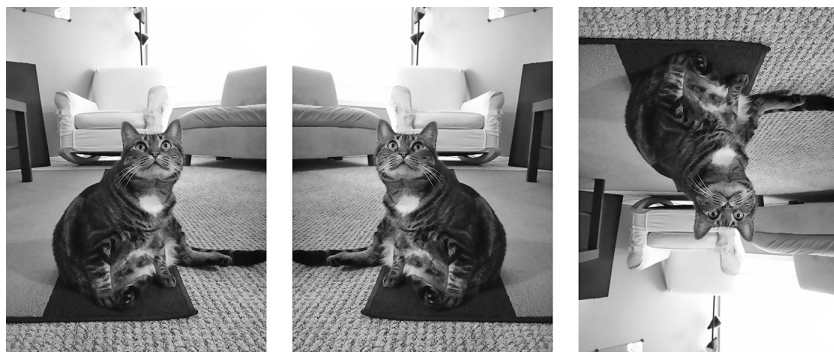


Figura 21.9. La imagen original (izquierda), la imagen volteada horizontalmente (centro) y la imagen volteada verticalmente (derecha).

Escribe esto en el intérprete de comandos interactivo:

```
> > >
```

```
cat_im.transpose(Image.FLIP_LEFT_RIGHT).save('horizontal_flip.png')
```

```
> > >
```

```
cat_im.transpose(Image.FLIP_TOP_BOTTOM).save('vertical_flip.png')
```

Al igual que `rotate()`, `transpose()` crea un nuevo objeto `Image`. Pasamos `Image.FLIP_LEFT_RIGHT` para voltear la imagen horizontalmente y guardamos el resultado en `horizontal_flip.png`. Para voltear la imagen verticalmente, pasamos `Image.FLIP_TOP_BOTTOM` y guardamos el resultado en `vertical_flip.png`.

Cambiar píxeles individuales

El método `getpixel()` puede recuperar el color de un píxel individual y `putpixel()` puede alterar ese color. Ambos métodos aceptan una tupla que representa las coordenadas `x` e `y` del píxel. El método

putpixel() toma un argumento adicional para el nuevo color del píxel, o como tupla RGBA de cuatro enteros o como tupla RGB de tres enteros. Introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > from PIL import Image
```

```
1 > > > im = Image.new('RGBA', (100, 100))
```

```
2 > > > im.getpixel((0, 0))
```

```
(0, 0, 0, 0)
```

```
3 > > > for x in range(100):
```

```
... for y in range(50):
```

```
. 4 im.putpixel((x, y), (210, 210, 210))
```

```
.
```

```
> > > from PIL import ImageColor
```

```
> > > for x in range(100):
```

```
5 . for y in range(50, 100):
```

```
. 6 im.putpixel((x, y), ImageColor.getcolor('darkgray', 'RGBA'))
```

```
.
```

```
> > > im.getpixel((0, 0))
```

```
(210, 210, 210, 255)
```

```
> > > im.getpixel((0, 50))
```

```
(169, 169, 169, 255)
```

```
> > > im.save('putPixel.png')
```

Hacemos una nueva imagen que es un cuadrado transparente de 100×100 . Llamar a getpixel() con coordenadas de esta imagen

devuelve (0, 0, 0, 0) porque es una imagen transparente 2. Para colorear sus píxeles, utilizamos bucles for anidados para pasar en bucle por los píxeles de la mitad superior de la imagen 3, pasando a putpixel() una tupla RGB que representa un gris claro 4.

Supongamos que queremos pintar la otra mitad de la imagen de gris oscuro, pero desconocemos la tupla RGB de ese color. El método putpixel() no acepta nombres de colores como 'darkgray', así que usamos ImageColor.getcolor() para obtener la tupla correspondiente 6. Pasamos en bucle por los píxeles de la mitad inferior de la imagen 5 y pasamos a putpixel() el valor de retorno de esta llamada, produciendo una imagen con una mitad gris clara y la otra gris oscura, como la figura 21.10. Llamamos a getpixel() con cualquiera de las coordenadas para confirmar que el color de un píxel determinado es el que esperamos. Por último, guardamos la imagen como putPixel.png.



Figura 21.10. La imagen putPixel.png.

Naturalmente, dibujar los píxeles de una imagen de uno en uno no es precisamente cómodo. Si tienes que dibujar formas, utiliza las funciones ImageDraw explicadas en el apartado “Dibujar en imágenes”.

Proyecto 16: Añadir un logo

Supón que tienes la aburrida tarea de redimensionar miles de imágenes y añadir un pequeño logo como marca de agua en la esquina de cada una. Si lo haces con programas de edición gráfica básicos como Paintbrush o Paint te puedes eternizar. Una aplicación de edición de imágenes mejor, como Photoshop, puede hacer procesamiento en lotes, pero ese software cuesta cientos de euros, así que vamos a hacer un *script* que se ocupe de esto. Imagina que la

figura 21.11 es el logo que hay que añadir en la esquina inferior derecha de cada imagen: un icono de gato negro con un borde blanco y un fondo transparente. Puedes usar tu propio logo o descargar el que se incluye en los recursos en línea de este libro.

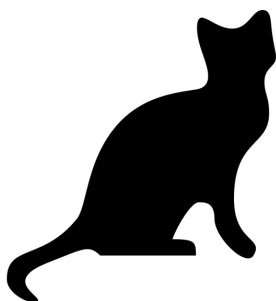


Figura 21.11. El logo que hay que añadir a la imagen.

Esto es lo que debería hacer el programa:

- Cargar la imagen del logotipo.
- Pasar en bucle por todos los archivos .png y .jpg del directorio de trabajo actual.
- Comprobar si la imagen tiene más de 300 píxeles de alto y de ancho.
- Si es así, reducir la altura o la anchura (lo que sea más grande) a 300 píxeles y reducir la otra dimensión proporcionalmente.
- Pegar la imagen del logo en la esquina.
- Guardar las imágenes modificadas en otra carpeta.

Esto significa que el código tendrá que hacer lo siguiente:

- Abrir el archivo catlogo.png como objeto Image.
- Pasar en bucle por las cadenas devueltas por `os.listdir('.')`.
- Obtener la anchura y la altura de la imagen a partir del atributo `size`.

- Calcular la nueva altura y anchura de la imagen redimensionada.
- Llamar al método `resize()` para redimensionar la imagen.
- Llamar al método `paste()` para pegar el logo en la esquina inferior derecha.
- Llamar al método `save()` para guardar los cambios, usando el nombre de archivo original.

Paso 1: abrir la imagen del logotipo

Abre una pestaña nueva en el editor de archivos, escribe el siguiente código y guárdalo como `resizeAndAddLogo.py`:

Redimensiona imágenes para encajar en un cuadrado de 300x300 con un logo en la esquina

```
import os
```

```
from PIL import Image
```

```
1 SQUARE_FIT_SIZE = 300
```

```
2 LOGO_FILENAME = 'catlogo.png'
```

```
3 logo_im = Image.open(LOGO_FILENAME)
```

```
4 logo_width, logo_height = logo_im.size
```

```
# PENDIENTE: Pasar en bucle por todos los archivos del directorio de trabajo actual.
```

```
# PENDIENTE: Comprobar si hay que redimensionar la imagen.
```

```
# PENDIENTE: Calcular la nueva anchura y altura para cambiar el tamaño.
```

```
# PENDIENTE: Redimensionar la imagen.
```

```
# PENDIENTE: Añadir el logo.
```

PENDIENTE: Guardar los cambios.

Al establecer las constantes `SQUARE_FIT_SIZE` 1 y `LOGO_FILENAME` 2 al principio del programa, es más fácil cambiar el programa después. Supongamos que el logo que vamos a añadir no es el icono del gato o que hay que reducir la dimensión más grande de las imágenes de salida a otro tamaño distinto de 300 píxeles. Podemos abrir el código fácilmente y cambiar esos valores solo una vez. También se podría establecer el valor de estas constantes aceptando argumentos de línea de comandos. Sin estas constantes, habría que buscar en el código todas las veces que aparezca 300 y 'catlogo.png' para sustituirlos por los nuevos valores.

El método `Image.open()` devuelve el objeto `Image` del logo 3. Por cuestiones de legibilidad, asignamos la anchura y la altura del logo a variables 4. El resto del programa es un esqueleto de comentarios PENDIENTE.

Paso 2: pasar en bucle por todos los archivos

Ahora tenemos que buscar todos los archivos .png y .jpg que haya en el directorio de trabajo actual. No queremos añadir la imagen del logo al propio logo, así que el programa debería omitir cualquier nombre de archivo que sea igual que `LOGO_FILENAME`. Añade lo siguiente al código:

Redimensiona imágenes para encajar en un cuadrado de 300x300 con un logo en la esquina

```
import os
```

```
from PIL import Image
```

--fragmento omitido--

```
os.makedirs('withLogo', exist_ok=True)
```

Pasa en bucle por todos los archivos del directorio de trabajo actual.

```
1 for filename in os.listdir('.')
```

```
2 if not (filename.endswith('.png') or  
filename.endswith('.jpg')) \
```

```
or filename = LOGO_FILENAME:
```

```
3 continue # Omite todos los archivos que no son imágenes y el  
archivo del logo.
```

```
4 im = Image.open(filename)
```

```
width, height = im.size
```

--fragmento omitido--

Primero, la llamada a `os.makedirs()` crea una carpeta `withLogo` donde se guardarán las imágenes originales, en lugar de sobrescribir los archivos originales. El argumento `exist_ok=True` evitará que `os.makedirs()` lance una excepción si `withLogo` ya existe. Mientras el código pasa en bucle por todos los archivos del directorio de trabajo actual 1, una declaración `if` larga comprueba los nombres que no terminan en `.png` o `.jpg` 2. Si encuentra alguno o si el archivo es el del propio logo, el bucle debería omitirlo y seguir con el siguiente 3. Si el nombre de archivo termina en `.png` o `.jpg` y no es el archivo con el logo, el código lo abre como objeto `Image` 4 y guarda su anchura y altura.

Paso 3: redimensionar las imágenes

El programa debería redimensionar la imagen solo si la anchura o la altura son mayores que `SQUARE_FIT_SIZE` (300 píxeles, en este caso), así que hay que poner el código de redimensionamiento dentro de una declaración `if` que compruebe las variables `width` y `height`. Añade el siguiente código al programa:

```
# Redimensiona imágenes para encajar en un cuadrado de 300x300  
con un logo en la esquina
```

```
import os
```

```
from PIL import Image
```


--fragmento omitido--

Comprueba si hay que redimensionar la imagen.

if width > SQUARE_FIT_SIZE and height > SQUARE_FIT_SIZE:

Calcula la nueva anchura y altura para cambiar el tamaño.

if width > height:

1 height = int((SQUARE_FIT_SIZE / width) * height)

width = SQUARE_FIT_SIZE

else:

2 width = int((SQUARE_FIT_SIZE / height) * width)

height = SQUARE_FIT_SIZE

Redimensiona la imagen.

print(f'Resizing {filename}...')

3 im = im.resize((width, height))

--fragmento omitido--

Si la imagen requiere cambio de tamaño, hay que averiguar si es por ser demasiado alta o demasiado ancha. Si width es mayor que height, el código debería reducir la altura en la misma proporción que la anchura 1. Esta proporción es el valor SQUARE_FIT_SIZE dividido entre la anchura actual, de modo que el código establece el nuevo valor de height en esta proporción multiplicada por el valor actual de height. Como el operador de división devuelve un valor flotante y resize() requiere que las dimensiones sean enteros, tenemos que acordarnos de convertir el resultado en un entero con la función int(). Por último, el código establecerá el nuevo valor de width en SQUARE_FIT_SIZE. Si height es mayor o igual que width, la proposición else realiza el mismo cálculo, pero intercambia las variables height y width 2. Cuando esas variables contienen las nuevas dimensiones, el código se las pasa al método resize() y

guarda el objeto Image devuelto 3.

Paso 4: añadir el logo y guardar los cambios

Tanto si se ha redimensionado la imagen como si no, hay que pegar el logo en la esquina inferior derecha. Dónde se insertará exactamente depende del tamaño de la imagen y del logotipo. La figura 21.12 muestra cómo calcular la posición de pegado. La coordenada izquierda en la que se pega el logo es la anchura de la imagen menos la anchura de logo, y la coordenada superior para pegar será la altura de la imagen menos la altura del logo. Una vez que el código pega el logo en la imagen, debería guardar el objeto Image modificado. Añade esto a tu programa:

```
# Redimensiona imágenes para encajar en un cuadrado de 300x300  
con un logo en la esquina
```

```
import os
```

```
from PIL import Image
```

```
--fragmento omitido--
```

```
# Comprueba si hay que redimensionar la imagen.
```

```
-- fragmento omitido --
```

```
# Añade el logo.
```

```
1 print(f'Adding logo to {filename}...')
```

```
2 im.paste(logo_im, (width - logo_width, height - logo_height),  
logo_im)
```

```
# Guarda los cambios.
```

```
3 im.save(os.path.join('withLogo', filename))
```

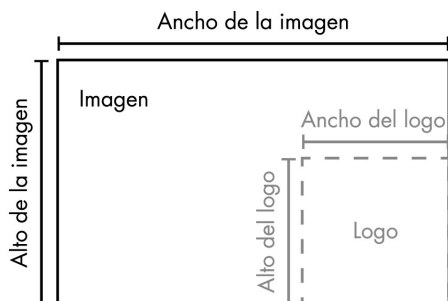


Figura 21.12. Las coordenadas izquierda y superior del logotipo son la anchura/altura de la imagen menos la anchura/altura del logo.

El nuevo código imprime un mensaje diciendo al usuario que se está añadiendo el logo 1, pega `logo_im` en `im` en las coordenadas calculadas 2 y guarda los cambios en un archivo en la carpeta `withLogo` 3. Al ejecutar este programa con `zophie.png` y otros archivos de imagen del directorio de trabajo actual, la salida debería ser así:

Resizing `zophie.png`...

Adding logo to `zophie.png`...

Resizing `zophie_xmas_tree.png`...

Adding logo to `zophie_xmas_tree.png`...

Resizing `me_and_zophie.png`...

Adding logo to `me_and_zophie.png`...

El programa convierte `zophie.png` en una imagen de 225×300 píxeles que queda como la figura 21.13.

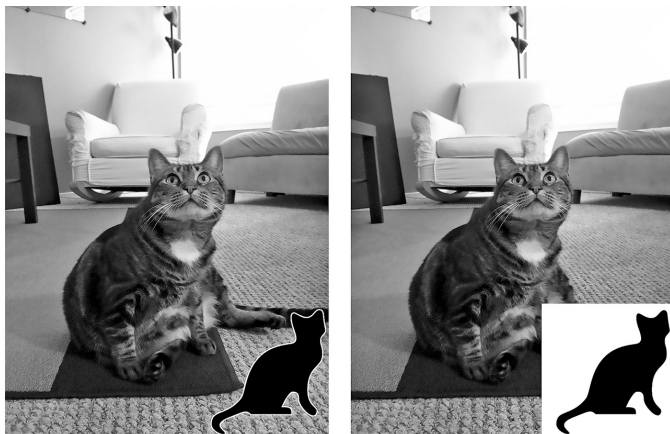


Figura 21.13. El programa ha cambiado el tamaño zophie.png y añadido el logo (izquierda). Si olvidas el tercer argumento, los píxeles transparentes del logo aparecerán como píxeles blancos (derecha).

Recuerda que el método `paste()` no pegará los píxeles transparentes a menos que pasemos `logo_im` como tercer argumento. Este programa puede redimensionar y “logificar” cientos de imágenes automáticamente en un par de minutos.

Ideas para programas similares

La capacidad de crear imágenes compuestas o modificar el tamaño de las imágenes en un lote puede ser útil para muchas aplicaciones. Podrías escribir programas similares para hacer lo siguiente:

- Añadir texto o una URL a imágenes.
- Añadir marcas de tiempo a imágenes.
- Copiar o mover imágenes a distintas carpetas en función de su tamaño.
- Añadir una marca de agua mayormente transparente a una imagen para evitar que otros la copien.

Dibujar en imágenes

Si tienes que dibujar líneas, rectángulos, círculos u otras formas simples en una imagen, utiliza el módulo ImageDraw de Pillow. Para ver un ejemplo, introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > from PIL import Image, ImageDraw  
> > > im = Image.new('RGBA', (200, 200), 'white')  
> > > draw = ImageDraw.Draw(im)
```

Primero, importamos Image e ImageDraw. A continuación, creamos una imagen blanca de 200×200 y la guardamos en im. Pasamos este objeto Image a la función ImageDraw.Draw() para recibir un objeto ImageDraw. Este objeto cuenta con muchos métodos para dibujar formas y texto. Guarda el nuevo objeto en una variable draw para poder usarla fácilmente en el siguiente ejemplo.

Formas

Los siguientes métodos ImageDraw dibujan distintos tipos de formas sobre la imagen. Los parámetros fill y outline para estos métodos son opcionales y serán por defecto blancos si no se especifica nada.

Puntos

El método point(xy, fill) dibuja píxeles individuales. El argumento xy representa una lista de los puntos que dibujar. La lista puede contener tuplas de coordenadas x e y, como [(x, y), (x, y), ..., o coordenadas x e y sin tuplas, como [x1, y1, x2, y2, ... El argumento fill colorea los puntos y puede ser una tupla RGBA o una cadena como 'red'. Este argumento es opcional. Aquí la palabra "punto" se refiere a un píxel, no a la unidad de tamaño de fuente.

Líneas

El método line(xy, fill, width) dibuja una línea o una serie de líneas. El argumento xy es una lista de tuplas, como [(x, y), (x, y), ..., o una lista de enteros, como [x1, y1, x2, y2, ... Cada punto es un punto de conexión en las líneas que estamos dibujando. El argumento

opcional fill especifica el color de las líneas como tupla RGBA o nombre de color. El argumento opcional width determina la anchura de las líneas y, si no se especifica nada, es 1 por defecto.

Rectángulos

El método `rectangle(xy, fill, outline, width)` dibuja un rectángulo. El argumento `xy` es una tupla de cuadro de la forma `(left, top, right, bottom)`. Los valores `left` y `top` especifican las coordenadas `x` e `y` de la esquina superior izquierda del rectángulo, mientras que `right` y `bottom` especifican las coordenadas de la esquina inferior derecha. El argumento opcional `fill` es el color de relleno del rectángulo. El argumento opcional `outline` es el color del contorno del rectángulo. El argumento opcional `width` representa la anchura de las líneas y, si no se especifica nada, es 1 por defecto.

Elipses

El método `ellipse(xy, fill, outline, width)` dibuja una elipse. Si la anchura y la altura de la elipse son idénticas, este método dibujará un círculo. El argumento `xy` es una tupla de cuadro `(left, top, right, bottom)` que representa una caja que contiene la elipse. El argumento opcional `fill` es el color de relleno de la elipse, y el argumento opcional `outline` es el color del contorno. El argumento opcional `width` representa la anchura de las líneas y, si no se especifica nada, es 1 por defecto.

Polígonos

El método `polygon(xy, fill, outline, width)` dibuja un polígono arbitrario. El argumento `xy` es una lista de tuplas, como `[(x, y), (x, y), ..., (x, y)]`, o enteros, como `[x1, y1, x2, y2, ..., xn, yn]`, que representa los puntos de conexión de los lados del polígono. El último par de coordenadas se conectará automáticamente con el primero. El argumento opcional `fill` es el color de relleno del polígono, y el argumento opcional `outline` es el color del contorno. El argumento opcional `width` representa la anchura de las líneas y, si no se especifica nada, es 1 por defecto.

Para practicar con estos métodos, escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > from PIL import Image, ImageDraw  
> > > im = Image.new('RGBA', (200, 200), 'white')  
> > > draw = ImageDraw.Draw(im)  
> > > draw.line([(0, 0), (199, 0), (199, 199), (0, 199), (0, 0)],  
fill='black') 1  
> > > draw.rectangle((20, 30, 60, 60), fill='blue') 2  
> > > draw.ellipse((120, 30, 160, 60), fill='red') 3  
> > > draw.polygon(((57, 87), (79, 62), (94, 85), (120, 90),  
(103, 113)), fill='brown') 4  
> > > for i in range(100, 200, 10): 5  
. draw.line([(i, 0), (200, i - 100)], fill='green')
```

Después de crear un objeto Image para una imagen blanca de 200 × 200, pasarlo a ImageDraw.Draw() para obtener un objeto ImageDraw y guardarlo en draw, podemos llamar a los metodos de dibujo con draw. Aquí, hemos puesto un contorno fino negro en los bordes de la imagen 1, un rectángulo azul cuya esquina superior izquierda está en (20, 30) y la inferior derecha en (60, 60) 2, una elipse roja definida por una caja desde (120, 30) hasta (160, 60) 3, un polígono marrón con cinco puntos 4, y un patrón de líneas verdes dibujadas con un bucle for 5. El archivo drawing.png resultante será como la figura 21.14 (aunque los colores no se imprimen en este libro).

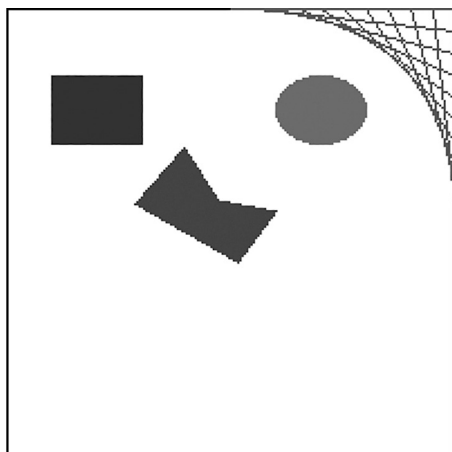


Figura 21.14. La imagen `drawing.png` resultante.

Podemos usar otros muchos métodos de dibujo de formas con objetos `ImageDraw`. Tienes toda la documentación a tu disposición en <https://pillow.readthedocs.io/en/latest/reference/ImageDraw.html>.

Texto

El objeto `ImageDraw` también tiene un método `text()` para dibujar texto en una imagen. Este método toma cuatro argumentos:

- `xy`: Una tupla de dos enteros que especifica la esquina superior izquierda del cuadro de texto.
- `text`: La cadena de texto que queremos escribir.
- `fill`: El color del texto.
- `font`: Un objeto `ImageFont` opcional para establecer la fuente y el tamaño del texto.

Antes de usar `text()` para dibujar texto en una imagen, vamos a ver el argumento opcional `font` con más detalle. Este argumento es un objeto `ImageFont` que podemos obtener ejecutando lo siguiente:

```
> > > from PIL import ImageFont
```


Una vez importado el módulo ImageFont de Pillow, accedemos a la fuente llamando a la función ImageFont.truetype(), que toma dos argumentos. El primero es una cadena que representa el archivo TrueType de la fuente, el archivo real de fuente que está en nuestro disco duro. Los archivos TrueType tienen la extensión .ttf y suelen estar en C:\Windows\Fonts en Windows, /Library/Fonts y /System/Library/Fonts en macOS, y /usr/share/fonts/truetype en Linux. No es necesario introducir estas rutas como parte de la cadena del archivo TrueType porque Pillow sabe buscar en esos directorios automáticamente, pero dará error si no encuentra la fuente especificada.

El segundo argumento de ImageFont.truetype() es un entero para el tamaño de fuente en puntos (que no en píxeles). Pillow crea imágenes PNG de 72 píxeles por pulgada por defecto y un punto es 1/72 de una pulgada. Para practicar, introduce esto en el intérprete de comandos interactivo:

```
> > > from PIL import Image, ImageDraw, ImageFont
> > > import os
1 > > > im = Image.new('RGBA', (200, 200), 'white')
2 > > > draw = ImageDraw.Draw(im)
3 > > > draw.text((20, 150), 'Hello', fill='purple')
4 > > > arial_font = ImageFont.truetype('arial.ttf', 32)
5 > > > draw.text((100, 150), 'Howdy', fill='gray',
font=arial_font)
> > > im.save('text.png')
```

Después de importar Image, ImageDraw, ImageFont y os, creamos un objeto Image para una nueva imagen blanca de 200 × 200 y un objeto ImageDraw a partir del objeto Image. Usamos text() para escribir Hello en (20, 150) en morado. No hemos pasado el cuarto argumento opcional en esta llamada, así que no hemos personalizado la fuente ni el tamaño.

A continuación, para especificar una fuente y un tamaño, llamamos a `ImageFont.truetype()`, pasándole el archivo `.ttf` de la fuente deseada, seguido de un entero para el tamaño 4. Guardamos el objeto `Font` devuelto en una variable, que pasamos a continuación al último argumento del método `text()`. La llamada al método dibuja `Howdy` en (100, 150) en gris, en Arial de 32 puntos 5. El archivo `text.png` será como la figura 21.15.

Si te interesa la creación de arte por ordenador con Python, te recomiendo *Learn Python Visually*, de Tristan Bunn (No Starch Press, 2021), o mi libro *The Recursive Book of Recursion* (No Starch Press, 2022).



Figure 21.15 shows a graphic with two lines of text. The first line, 'Hello', is rendered in a small, black, monospaced font. The second line, 'Howdy', is rendered in a much larger, gray, sans-serif font.

Figura 21.15. La imagen `text.png` resultante.

Copiar y pegar imágenes en el portapapeles

Igual que el módulo de terceros `pyperclip` nos permite copiar y pegar cadenas de texto en el portapapeles, el módulo `pyperclipimg` puede copiar objetos `Image` de `Pillow`. Para instalar `pyperclipimg`, sigue las instrucciones del apéndice A.

La función `pyperclipimg.copy()` coge un objeto `Image` de `Pillow` y lo pone en el portapapeles del sistema operativo. Después, podemos pegarlo en un programa de procesamiento de gráficos o imágenes como `MS Paint`. La función `pyperclipimg.paste()` devuelve el contenido de la imagen del portapapeles como un objeto `Image`. Con `zophie.png` en el directorio de trabajo actual, introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > from PIL import Image
```

```
> > > im = Image.open('zophie.png')
> > > import pyperclipimg
> > > pyperclipimg.copy(im)

> > > pasted_im = pyperclipimg.paste() # Ahora copia una
nueva imagen en el portapapeles.

> > > # Copia el contenido del portapapeles en un programa
gráfico.

> > > pasted_im.show() # Muestra la imagen del portapapeles.
```

En este código, primero abrimos la imagen `zophie.png` como un objeto `Image` y, después, lo pasamos a `pyperclipimg.copy()` para copiarlo en el portapapeles. Podemos comprobar que la copia ha funcionado pegando la imagen en un programa gráfico. A continuación, copiamos una nueva imagen de un programa gráfico o haciendo clic con el botón secundario del ratón en una imagen del navegador y copiándola. Llamar a `pyperclipimg.paste()` devuelve esta imagen como un objeto `Image` en la variable `pasted_im`. Podemos comprobar que el pegado ha funcionado mirándolo con `pasted_im.show()`.

El módulo `pyperclipimg` puede ser útil como método para permitir a los usuarios introducir y extraer datos de imagen con nuestros programas de Python.

Crear gráficos con Matplotlib

Dibujar tus propios gráficos con Pillow es posible, pero lleva mucho trabajo. La biblioteca Matplotlib crea una amplia variedad de gráficos para usar en publicaciones profesionales. En este capítulo, crearemos gráficos de líneas básicos, gráficos de barras, diagramas de dispersión y gráficos circulares, pero Matplotlib también puede crear gráficos 3D más complejos. Puedes consultar la documentación completa en <https://matplotlib.org>. Instala Matplotlib siguiendo las instrucciones del apéndice A.

Empezaremos creando un gráfico de líneas 2D con dos ejes, x e y . Un gráfico de líneas es ideal para mostrar cambios en una medida a lo largo del tiempo. En la documentación de Matplotlib, los términos “plot”, “graph” y “chart” se usan a menudo de manera intercambiable para referirse a los gráficos, y el término “figure” hace referencia a la ventana que contiene uno o más gráficos. Introduce esto en el intérprete de comandos interactivo:

```
> > > import matplotlib.pyplot as plt 1
> > > x_values = [0, 1, 2, 3, 4, 5]
> > > y_values1 = [10, 13, 15, 18, 16, 20]
> > > y_values2 = [9, 11, 18, 16, 17, 19]
> > > plt.plot(x_values, y_values1) 2
[<matplotlib.lines.Line2D object at 0x000002501D9A7D10>]
> > > plt.plot(x_values, y_values2)
[<matplotlib.lines.Line2D object at 0x00000250212AC6D0>]
> > > plt.savefig('linegraph.png') # Guarda el gráfico como un
archivo de imagen.
> > > plt.show() # Abre una ventana con el gráfico.
> > > plt.show() # No hace nada.
```

Importamos matplotlib.pyplot con el nombre plt 1 para que sea más fácil introducir sus funciones. A continuación, para trazar los puntos de datos en una figura bidimensional, tenemos que llamar a la función plt.plot(). Primero, guardamos una lista de enteros o flotantes en x_values para el eje x y, después, guardamos una lista de enteros o flotantes en y_values1 para el eje y 2. Los primeros valores de las listas de los ejes x e y están asociados entre sí, los segundos valores de las listas también se asocian entre sí, y así sucesivamente. Después de llamar a plt.plot() con estos valores, la llamamos una segunda vez con x_values e y_values2 para añadir una segunda línea al gráfico.

Matplotlib seleccionará automáticamente los colores de las líneas y un tamaño adecuado para el gráfico. Podemos guardar el gráfico predeterminado como imagen llamando a `plt.savefig('linegraph.png')`.

Matplotlib tiene una función de vista previa que muestra el gráfico en una ventana, igual que Pillow tiene el método `show()` para la vista previa de objetos Image. Llama a `plt.show()` para abrir el gráfico en una ventana. Quedará como la figura 21.16.

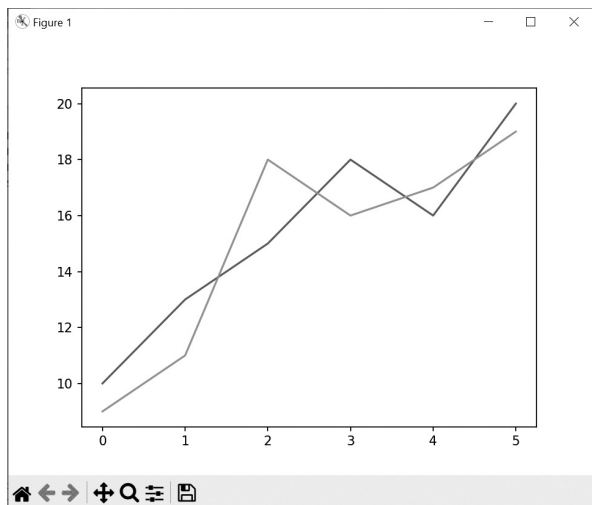


Figura 21.16. Un gráfico de líneas mostrado con `plt.show()`.

La ventana que crea `plt.show()` es interactiva, podemos mover el gráfico y alejar o acercar la imagen. El icono de la casa, en la esquina inferior izquierda, reestablece la vista, y el icono del disquete permite guardar el gráfico como una imagen. Si estás experimentando con datos, `plt.show()` es una herramienta de visualización cómoda. La llamada a la función `plt.show()` se bloqueará y no volverá hasta que el usuario cierre esta ventana.

Al cerrar la ventana que crea el método `plt.show()`, también reestableces los datos del gráfico. Llamar a `plt.show()` una segunda vez no hace nada o muestra una ventana vacía. Tendrás que llamar a `plt.plot()` y otras funciones relacionadas de nuevo para volver a crear el gráfico. Para guardarlo en un archivo de imagen, debes llamar a `plt.savefig()` antes de llamar a `plt.show()`.

Para crear un diagrama de dispersión con los mismos datos, pasa los valores de los ejes x e y a la función `plt.scatter()`:

```
> > > import matplotlib.pyplot as plt

> > > x_values = [0, 1, 2, 3, 4, 5]

> > > y_values1 = [10, 13, 15, 18, 16, 20]

> > > y_values2 = [9, 11, 18, 16, 17, 19]

> > > plt.scatter(x_values, y_values1)

<matplotlib.collections.PathCollection object at
0x000000250212CBAD0>

> > > plt.scatter(x_values, y_values2)

<matplotlib.collections.PathCollection object at
0x0000002502132DC10>

> > > plt.savefig('scatterplot.png')

> > > plt.show()
```

Cuando llamamos a `plt.show()`, Matplotlib muestra el gráfico de la figura 21.17. El código para crear un diagrama de dispersión es idéntico al que crea un gráfico de líneas, excepto en la llamada a la función.

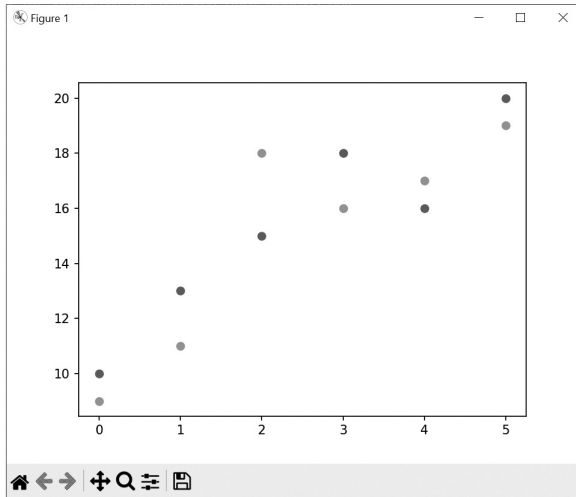


Figura 21.17. Un diagrama de dispersión mostrado con `plt.show()`.

Si comparas este gráfico con el gráfico de líneas de la figura 21.16, verás que los datos son los mismos, aunque los diagramas de dispersión usan puntos en vez de líneas conectadas.

Gráficos de barras y circulares

Vamos a crear un gráfico de barras básico con Matplotlib. Los gráficos de barras son útiles para comparar el mismo tipo de datos en distintas categorías. A diferencia de los gráficos de líneas, el orden de las categorías no es importante aquí, aunque suelen aparecer en orden alfabético. Introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > import matplotlib.pyplot as plt
> > > categories = ['Cats', 'Dogs', 'Mice', 'Moose']
> > > values = [100, 200, 300, 400]
> > > plt.bar(categories, values)
<BarContainer object of 4 artists>
> > > plt.savefig('bargraph.png')
```

```
> > > plt.show()
```

Este código crea el gráfico de barras que muestra la figura 21.18. Pasamos las categorías a una lista en el eje *x* como primer argumento de `plt.bar()` y los valores de cada categoría como el segundo argumento de lista.

Recuerda que cerrar la ventana de `plt.show()` reestablece los datos del gráfico.

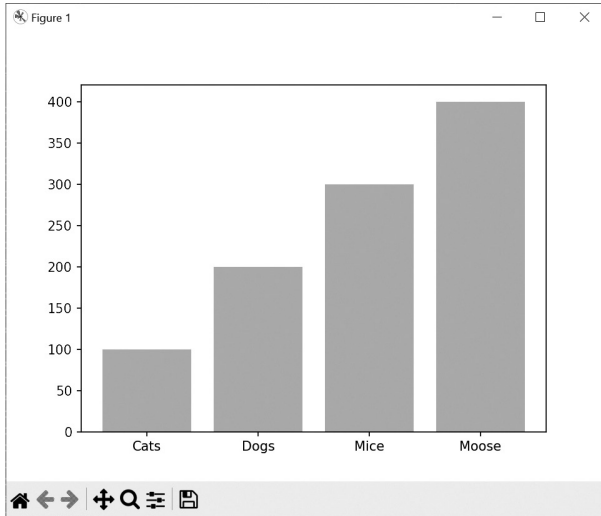


Figura 21.18. Un gráfico de barras mostrado con `plt.show()`.

Para crear un gráfico circular, llama a la función `plt.pie()`. En lugar de categorías y valores, un gráfico circular tiene etiquetas y secciones. Escribe esto en el intérprete de comandos interactivo:

```
> > > import matplotlib.pyplot as plt
```

```
> > > slices = [100, 200, 300, 400] # El tamaño de cada  
sección.
```

```
> > > labels = ['Cats', 'Dogs', 'Mice', 'Moose'] # El nombre de  
cada sección.
```

```
> > > plt.pie(slices, labels=labels, autopct='%.1f%%')
```


([<matplotlib.patches.Wedge object at 0x00000218F32BA950> ,

--fragmento omitido--

```
> > > plt.savefig('piechart.png')
```

```
> > > plt.show()
```

Cuando llamamos a call `plt.show()` para crear un gráfico circular, Matplotlib lo muestra en una ventana, como en la figura 21.19. La función `plt.pie()` acepta una lista de tamaños y una lista de etiquetas para cada sección.

El argumento `autopct` especifica la precisión de la etiqueta de porcentaje para cada sección. El argumento es una cadena especificadora de formato: la cadena `'%.1f%%'` especifica que el número debería mostrar un dígito después del punto decimal. Si dejamos este argumento fuera de la llamada a la función, el gráfico circular no mostrará el texto del porcentaje.

Matplotlib selecciona automáticamente los colores para cada sección, pero se puede personalizar este comportamiento, además de otros muchos aspectos de los gráficos que creemos.

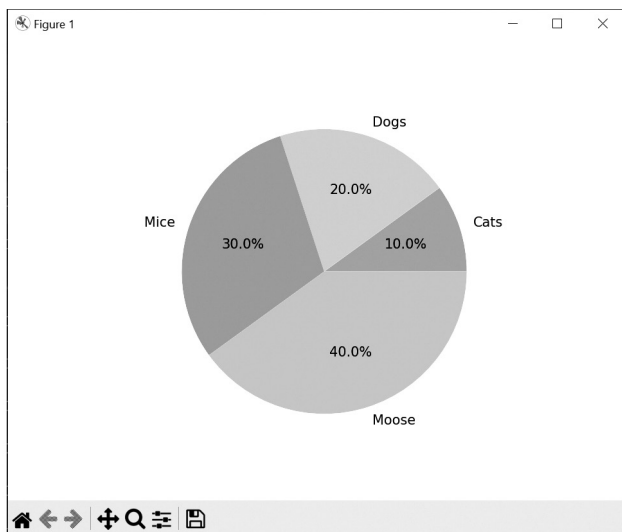


Figura 21.19. Un gráfico circular mostrado con `plt.show()`.

Los gráficos creados en los apartados anteriores son bastante sencillos. Matplotlib cuenta con un gran número de funciones adicionales con las que se podría escribir otro libro, así que aquí veremos solo los componentes más habituales. Vamos a añadir marcadores de puntos de datos, colores personalizados y etiquetas a nuestros gráficos. Introduce esto en el intérprete de comandos interactivo:

```
> > > import matplotlib.pyplot as plt
```

```
> > > x_values = [0, 1, 2, 3, 4, 5]
```

```
> > > y_values1 = [10, 13, 15, 18, 16, 20]
```

```
> > > y_values2 = [9, 11, 18, 16, 17, 19]
```

```
1 > > > plt.plot(x_values, y_values1, marker='o', color='b',  
label='Line 1')
```

```
[ <matplotlib.lines.Line2D object at 0x000001BC339D2F90 > ]
```

```
> > > plt.plot(x_values, y_values2, marker='s', color='r',  
label='Line 2')
```

```
[ <matplotlib.lines.Line2D object at 0x000001BC339D1A90 > ]
```

```
2 > > > plt.legend()
```

```
<matplotlib.legend.Legend object at 0x000001BC20915B90 >
```

```
3 > > > plt.xlabel('X-axis Label')
```

```
Text(0.5, 0, 'X-axis Label')
```

```
> > > plt.ylabel('Y-axis Label')
```

```
Text(0, 0.5, 'Y-axis Label')
```

```
> > > plt.title('Graph Title')
```

```
Text(0.5, 1.0, 'Graph Title')
```

```
4 >>> plt.grid(True)
```

```
>>> plt.show()
```

Después de ejecutar este código, Matplotlib muestra una ventana como la de la figura 21.20. Contiene el mismo gráfico de líneas que hemos creado antes, pero hemos añadido argumentos `marker`, `color` y `label` en las llamadas a la función `plt.plot()` 1. El argumento `marker` crea un punto para cada dato de la línea. Un valor `'o'` hace que el punto sea un círculo en forma de O, mientras que `'s'` lo hace cuadrado. Los argumentos de color `'b'` y `'r'` hacen la línea azul y roja, respectivamente. Ponemos a cada línea una etiqueta que usar en la leyenda creada llamando a `plt.legend()` 2.

También creamos etiquetas para los ejes `x` e `y`, para todo el gráfico, llamando a `plt.xlabel()`, `plt.ylabel()` y `plt.title()` 3 y pasando el texto de las etiquetas como cadenas. Por último, pasar `True` a `plt.grid()` 4 habilita una cuadrícula con líneas a lo largo de los ejes `x` e `y`.

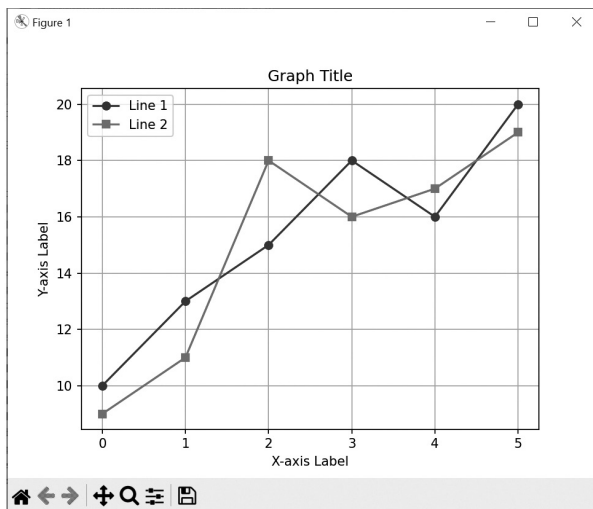


Figura 21.20. El gráfico de líneas de ejemplo con componentes adicionales.

Esto es solo una pequeña muestra de las funciones que ofrece Matplotlib. Para saber más, consulta la documentación en línea.

Resumen

Las imágenes consisten en una colección de píxeles que tienen un valor RGBA para su color y un conjunto de coordenadas x e y que representa su ubicación. Dos formatos de imagen habituales son JPEG y PNG. Pillow puede manejar ambos formatos y otros muchos. Cuando un programa carga una imagen en un objeto Image, sus dimensiones de altura y anchura se guardan como una tupla de dos enteros en el atributo size. Los objetos del tipo de datos Image también tienen métodos para manipulaciones de imágenes habituales: crop(), copy(), paste(), resize(), rotate() y transpose(). Para guardar el objeto Image en un archivo de imagen, llama al método save().

Si quieres que tu programa dibuje formas en una imagen, utiliza los métodos ImageDraw para dibujar puntos, líneas, rectángulos, elipses y polígonos. El módulo también tiene métodos para dibujar texto con la fuente y el tamaño que elijas.

Aunque la biblioteca Pillow permite dibujar formas y píxeles individuales, es más fácil generar gráficos con la biblioteca Matplotlib. Podemos crear gráficos de barras, de líneas y circulares con los ajustes predeterminados de Matplotlib o especificar personalizaciones. El método show() muestra el gráfico en la pantalla para una vista previa y el método save() genera archivos de imagen que podemos incluir en documentos u hojas de cálculo. Encontrarás más información sobre estas funciones en la documentación en línea de la biblioteca.

Aunque hay aplicaciones avanzadas (y caras), como Photoshop, que ofrecen funciones de procesamiento por lotes, podemos usar *scripts* de Python para hacer muchas de las mismas modificaciones de manera gratuita. En los capítulos anteriores, has escrito programas de Python para manejar archivos de texto simple, hojas de cálculo, PDF y otros formatos. ¡Con Pillow, has aumentado tus poderes de programación para procesar también imágenes!

Preguntas prácticas

1. ¿Qué es un valor RGBA?

2. ¿Cómo podemos obtener el valor RGBA de 'CornflowerBlue' en el módulo Pillow?
3. ¿Qué es una tupla de cuadro?
4. ¿Qué función devuelve un objeto Image para, digamos, un archivo de imagen llamado zophie.png?
5. ¿Cómo podemos averiguar la anchura y la altura de la imagen de un objeto Image?
6. ¿A qué método llamarías para obtener el objeto Image para el cuarto inferior izquierdo de una imagen de 100×100 ?
7. Después de hacer cambios en un objeto Image, ¿cómo los guardarías como archivo de imagen?
8. ¿Qué módulo contiene el código para dibujar formas en Pillow?
9. Los objetos Image no tienen métodos de dibujo. ¿Qué tipo de objeto sí los tiene? ¿Cómo se obtiene ese tipo de objeto?
10. ¿Qué funciones de Matplotlib crean un gráfico de líneas, un diagrama de dispersión, un gráfico de barras y un gráfico circular?
11. ¿Cómo se puede guardar un gráfico de Matplotlib como una imagen?
12. ¿Qué hace la función `plt.show()` y por qué no podemos llamarla dos veces seguidas?

Programas de práctica

Para practicar, escribe programas que realicen las siguientes tareas.

Creador de mosaicos

Escribe un programa que produzca un mosaico de imágenes a partir de una misma imagen, como el de caras de gato de la figura 21.6. El programa debería tener una función `make_tile()` con tres argumentos: una cadena con el nombre de archivo de la imagen, un

entero para cuántas veces debería aparecer horizontalmente y un entero para cuántas veces debería aparecer verticalmente. La función `make_tile()` debería devolver un objeto `Image` más grande de la imagen en mosaico. Utiliza los métodos `paste()` como parte de esta función.

Por ejemplo, si `zophie_the_cat.jpg` fuese una imagen de 20×50 píxeles, llamar a `make_tile('zophie_the_cat.jpg', 6, 10)` debería devolver una imagen de 120×500 con 60 teselas en total. Como extra, intenta voltear o rotar aleatoriamente la imagen para el mosaico antes de pegarla en la imagen más grande. Este creador de mosaicos funciona mejor con imágenes más pequeñas como teselas. A ver qué arte abstracto creas con este código.

Identificar carpetas de fotos en el disco duro

Tengo la mala costumbre de pasar archivos de mi cámara digital a carpetas temporales en alguna parte del disco duro y olvidarme de ellas. Estaría bien escribir un programa que pudiera escanear todo el disco duro para localizar esas carpetas olvidadas de fotos.

Escribe un programa que pase por todas las carpetas del disco duro y busque posibles carpetas de fotos. Naturalmente, primero hay que definir lo que se considera una “carpeta de fotos”, así que vamos a suponer que es cualquier carpeta donde más de la mitad de los archivos son fotos. ¿Y cómo definimos qué archivos son fotos? Primero, un archivo de fotos debe tener la extensión `.png` o `.jpg`. Además, las fotos son imágenes grandes, la anchura y la altura del archivo de foto deben ser mayores que 500 píxeles. Es una apuesta segura, ya que la mayoría de las fotos hechas con cámaras digitales tienen varios miles de píxeles de alto y ancho.

Aquí tienes una pequeña ayuda, el esqueleto del programa:

```
# Importa módulos y escribe comentarios para describir el
programa.
```

```
for folder_name, subfolders, filenames in os.walk('C:\\\\'):
```

```
    num_photo_files = 0
```

```
num_non_photo_files = 0

for filename in filenames:

    # Comprueba si la extensión de archivo no es .png ni .jpg.
    if PENDIENTE:

        num_non_photo_files += 1

    continue # Salta al siguiente nombre de archivo.

    # Abre la imagen con Pillow.

    # Comprueba si la anchura y la altura son mayores que 500.
    if PENDIENTE:

        # La imagen es lo bastante grande para considerarse una foto.
        num_photo_files += 1

    else:

        # La imagen es demasiado pequeña para ser una foto.
        num_non_photo_files += 1

    # Si más de la mitad de los archivos eran fotos,
    # imprime la ruta absoluta a la carpeta.

    if PENDIENTE:

        print(PENDIENTE)
```

Al ejecutar el programa, debería aparecer en la pantalla la ruta absoluta a cualquier carpeta de fotos.

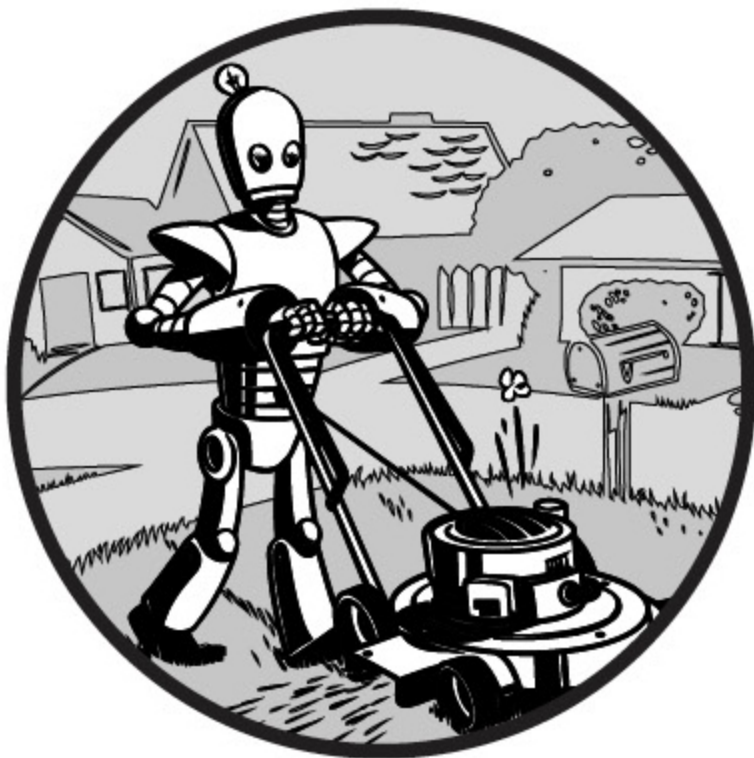
Crear tarjetas de asiento personalizadas

En un programa de práctica del capítulo 17, creaste invitaciones personalizadas para una lista de invitados en un archivo de texto

simple. Como proyecto complementario, usa Pillow para crear imágenes que sirvan como tarjetas de asiento para esos invitados. Para cada uno de los invitados recogidos en el archivo `guests.txt` de los recursos en línea del libro, genera una imagen con el nombre del invitado y una decoración floral. En los recursos del libro también tienes una imagen floral de dominio público.

Para asegurarte de que todas las tarjetas tienen el mismo tamaño, añade un rectángulo negro a los bordes de la imagen de la invitación; de ese modo, al imprimirla, tendrás una guía para recortar. Los archivos PNG que produce Pillow están configurados a 72 píxeles por pulgada, así que una tarjeta de 4×5 pulgadas (10×13 cm aproximadamente) requerirá una imagen de 288×360 píxeles.

Reconocer texto en imágenes



El reconocimiento de texto, más formalmente denominado reconocimiento óptico de caracteres (OCR), es la extracción de texto de una imagen. Python cuenta con una amplia colección de métodos de cadenas y expresiones regulares para procesar texto, pero todos requieren que primero se introduzca el texto como una cadena. Los programas pueden utilizar el OCR para, por ejemplo, reconocer los nombres de las señales de tráfico y los datos escritos en los cheques depositados en un cajero automático o para escanear recibos y crear copias electrónicas.

Al igual que la conversión de texto a voz o el reconocimiento de

voz, el OCR implica el uso de técnicas informáticas avanzadas, pero los módulos de Python ocultan estos detalles, lo que facilita su uso. Este capítulo aborda PyTesseract, el paquete de Python que funciona con el motor OCR de código abierto Tesseract. También veremos la aplicación gratuita NAPS2, que Python puede ejecutar para aplicar el OCR de Tesseract a archivos PDF.

Instalar Tesseract y PyTesseract

Para trabajar con PyTesseract, tienes que instalar el software de OCR gratuito Tesseract en tu ordenador Windows, macOS o Linux siguiendo las instrucciones de este apartado. También puedes instalar los paquetes de idiomas para lenguas distintas del inglés. Después, instala el paquete PyTesseract para que tus *scripts* de Python puedan interactuar con Tesseract.

Windows

En Windows, abre tu navegador en <https://github.com/UB-Mannheim/tesseract/wiki> y sigue las instrucciones de la página para descargar el instalador más reciente. A continuación, haz doble clic en él para instalar Tesseract.

Por defecto, Tesseract reconoce texto en inglés. Durante la instalación, puedes seleccionar las casillas Additional script data (download) y Additional language data (download) para que Tesseract reconozca otros alfabetos e idiomas. Instalar todos los idiomas añade unos 600 MB al tamaño de la instalación. Estos paquetes de idioma tienen nombres de archivo que identifican el idioma y una extensión .traineddata, por ejemplo, jpn.traineddata es para el japonés. Como alternativa, puedes seleccionar las casillas para idiomas individuales para ahorrar espacio.

Una vez terminada la instalación, añade la carpeta C:\Program Files\Tesseract-OCR (o la carpeta en la que hayas instalado Tesseract) a la variable de entorno PATH para que PyTesseract pueda acceder al programa tesseract.exe. El capítulo 12 explica cómo modificar esta variable de entorno.

macOS

El gestor de paquetes Homebrew puede instalar Tesseract en macOS. Ve a <https://docs.brew.sh> para instalar Homebrew y, después, instala Tesseract abriendo una ventana del terminal y ejecutando `brew install tesseract`. A continuación, ejecuta `brew install tesseract-lang` para instalar los paquetes de idiomas.

Linux

Para instalar Tesseract en Linux, abre una ventana del terminal y ejecuta `sudo apt install tesseract-ocr`. Tendrás que introducir la contraseña de administrador para ejecutar este comando. Para instalar los paquetes para todos los idiomas, ejecuta `sudo apt install tesseract-ocr-all` desde el terminal. Para instalar solo los paquetes de idiomas que quieras, sustituye `all` por un código ISO 639 de tres caracteres, como `fra` para francés, `deu` para alemán, `jpn` para japonés o `spa` para español.

PyTesseract

Tras instalar el motor de OCR Tesseract, puedes instalar la última versión de PyTesseract siguiendo las instrucciones del apéndice A. PyTesseract también instala la biblioteca de imágenes Pillow.

Fundamentos del OCR

Con PyTesseract y la biblioteca de imágenes Pillow, podemos extraer texto de una imagen con cuatro líneas de código. Hay que importar las bibliotecas PyTesseract y Pillow, abrir la imagen con la función `Image.open()` y, por último, pasar la imagen abierta a la función `tesseract.image_to_string()`. Vamos a ver un ejemplo básico: extraer texto de una captura de pantalla de la introducción de mi libro *The Big Book of Small Python Projects* (No Starch Press, 2021). Descarga la imagen `ocr-example.png` en los recursos en línea del libro y, después, introduce lo siguiente en el intérprete de comandos interactivo para abrir la imagen con Pillow y escanearla con Tesseract:

```
> > > import pytesseract as tess
```

```
> > > from PIL import Image
```

```
> > > img = Image.open('ocr-example.png')
```

```
> > > text = tess.image_to_string(img)
```

```
> > > print(text)
```

This book provides you with practice examples of how programming

concepts are applied, with a collection of over 80 games, simulations, and dig-

ital art programs. These aren't code snippets; they're full, runnable Python

programs. You can copy their code to become familiar with how they work,

experiment with your own changes, and then attempt to re-create them on

your own as practice. After a while, you'll start to get ideas for your own pro-

grams and, more importantly, know how to go about creating them.

--fragmento omitido--

Convertir texto de una imagen en una cadena requiere algoritmos sofisticados, pero Python lo hace accesible ¡con cuatro líneas de código!

Preprocesar una imagen

El texto de la imagen del apartado anterior se extrajo casi a la perfección en una cadena de Python. Sin embargo, el OCR tiene limitaciones. A diferencia de las imágenes generadas por ordenador, como las capturas de pantalla, el papel escaneado o fotocopiado puede tener taras y las fotografías de escenas reales son demasiado complicadas para extraer texto. Por ejemplo, no podemos hacer una foto a la trasera de un coche y esperar que Tesseract extraiga de ahí el número de matrícula. Antes, habría que recortar la imagen

alrededor de la matrícula, e incluso así podría ser ilegible. Por ese motivo, Tesseract está pensado para documentos impresos más que fotos o texto manuscrito.

Incluso aunque trabajes con capturas de pantalla, piensa siempre que el texto del OCR es imperfecto y necesita corrección. En concreto, podrías encontrar problemas como estos:

- La cadena mantiene guiones para separar palabras al final de línea (en nuestro ejemplo, en “dig-” y “ital” o “pro-” y “grams”).
- La cadena no conserva información de fuente ni tamaño.
- El espacio en blanco de la cadena podría no coincidir con el texto.
- La cadena puede haber escaneado incorrectamente caracteres, por ejemplo confundiendo una j minúscula con una i.
- Si la imagen contiene tablas o múltiples columnas de texto, la cadena puede mezclar el texto y desordenarlo.

Presta especial atención a los errores en números, ya que pueden ser más difíciles de detectar que las palabras mal escritas.

Tesseract da ciertos pasos de procesamiento para mitigar algunos problemas, pero puedes mejorar su precisión usando un programa de edición de imágenes para hacer lo siguiente:

- No escanees imágenes con múltiples columnas; pon cada columna de texto en una imagen aparte.
- Usa solo texto escrito a máquina, no a mano.
- Usa fuentes convencionales, no cursivas ni estilizadas.
- Rota la imagen para que las líneas de texto estén perfectamente alineadas y no ligeramente inclinadas.
- Usa texto oscuro sobre fondo claro, no texto blanco sobre fondo oscuro.
- Elimina cualquier borde oscuro alrededor de la imagen.

- Añade un pequeño borde blanco si el texto llega hasta el extremo de la imagen.
- Ajusta el brillo y el contraste de la imagen para que el texto destaque sobre el fondo.
- Elimina el “ruido” para limpiar la imagen antes de escanear.

Puedes hacer algunos de estos pasos automáticamente con Python empleando la biblioteca OpenCV. Echa un vistazo al *post* “Preprocessing Images for OCR with Python and OpenCV”, en <https://autbor.com/preprocessingocr> para más ejemplos.

Corregir errores con modelos de lenguaje de gran tamaño

Los tipos de errores que suelen cometer los algoritmos de OCR implican espacios y caracteres individuales. Usar un algoritmo de corrección ortográfica no detectará los errores del OCR: señalará los caracteres bien reconocidos de palabras mal escritas en la imagen original pero no verá los errores que producen palabras bien escritas. Identificar este tipo de errores requiere entender el contexto y tener sentido común para saber qué deberían ser los caracteres.

Ese es exactamente el tipo de problema que pueden resolver los modelos de lenguaje de gran tamaño (LLM), como ChatGPT, Gemini y LLaMA. Por ejemplo, observa la figura 22.1, un escaneo de la novela *Frankenstein*, de Mary Shelley. Esta página en concreto se imprimió en 1831, así que el papel está arrugado y amarillento y la tinta no está igual en todos los caracteres. Puedes descargar [frankenstein.png](#) en los recursos en línea del libro.

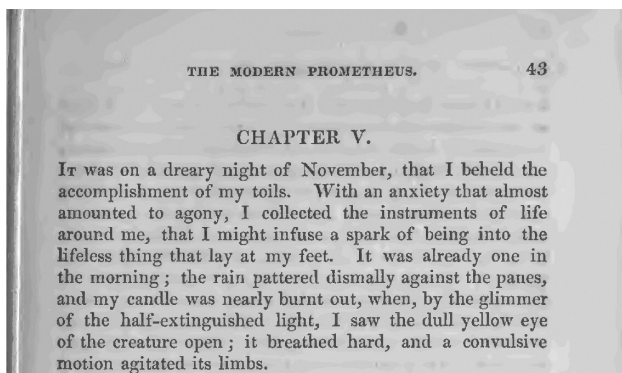


Figura 22.1. La parte superior de una página escaneada de una edición de 1831 de *Frankenstein*, de Mary Shelley.

Sin preprocesamiento, Tesseract identifica esto:

```
> > > import pytesseract as tess
> > > from PIL import Image
> > > img = Image.open('frankenstein.png')
> > > text = tess.image_to_string(img)
> > > print(text)
```

THE MODERN PROMETHEUS. 43

CHAPTER V.

Iv was on a dreary night of November, that I beheld the accomplishment of my toils. With an anxiety that almost amounted to agony, I collected the instruments of life around me, that I might infuse a spark of being into the lifeless thing that lay at my feet. It was already one in the morning; the rain pattered dismally against the panes,

and my candle was nearly burnt out, when, by the glimmer
of the half-extinguished light, I saw the dull yellow eye
of the creature open; it breathed hard, and a convulsive
Motion agitated its limbs.

Hace un trabajo bastante bueno, pero produce caracteres incorrectos al principio del párrafo, 'Iv' en vez de 'IT', y en la última línea, 'Motion' en lugar de 'motion'. (El resultado exacto puede variar a medida que los paquetes de lenguas se actualicen).

Para corregir estos problemas, prueba a dar a un LLM el siguiente *prompt* de ejemplo con el texto de salida del escaneo de *Frankenstein*:

Lo siguiente es el texto obtenido de un escaneo OCR. Corrige cualquier espacio, carácter faltante o carácter reconocido incorrectamente. No corrijas errores ortográficos o gramaticales que existan en el texto original. Coloca los párrafos en una sola línea y deshaz las palabras separadas por guiones que se dividen al final de una línea. Solo proporciona el texto corregido sin explicar qué es el OCR ni ningún otro prefacio. Aquí está el texto:

THE MODERN PROMETHEUS. 43

CHAPTER V.

Iv was on a dreary night of November, that I beheld the...

Con este *prompt*, el LLM produjo esta salida:

THE MODERN PROMETHEUS. 43

CHAPTER V.

It was on a dreary night of November that I beheld the
accomplishment of

my toils. With an anxiety that almost amounted to agony, I
collected the

instruments of life around me, that I might infuse a spark of being into

the lifeless thing that lay at my feet. It was already one in the morning;

the rain pattered dismally against the panes, and my candle was nearly burnt

out, when, by the glimmer of the half-extinguished light, I saw the dull

yellow eye of the creature open; it breathed hard, and a convulsive motion

agitated its limbs.

El LLM corrigió ‘Iv’ por ‘It’ y ‘Motion’ por ‘motion’. También eliminó las palabras con guion al final de las líneas para quitar los saltos de línea. Así es más fácil copiar y pegar el texto en un documento de Word o en un correo electrónico. Para automatizar este proceso, la mayoría de los LLM cuentan con API, de modo que nuestros programas pueden enviar *prompts* directamente y recibir respuestas. A menos que ejecutes un LLM en tu ordenador en local (lo cual no entra en el ámbito de este libro), tendrás que registrarte para estos servicios LLM en línea. La suscripción puede ser gratuita o de pago.

Recuerda siempre que los LLM tienden a un exceso de confianza en sí mismos, así que comprueba sus resultados. El texto que devuelven podría haberse dejado algún error, corregido el tipo de error equivocado o incluso introducido otros errores. Sigue haciendo falta un humano para revisar la salida de la máquina. (Y puede que un segundo humano para revisar el trabajo del primer humano, ya que los humanos también cometen errores a menudo.)

Reconocer texto en idiomas distintos del inglés

Tesseract asume que el texto que está escaneando está en inglés por defecto, pero podemos especificar otros idiomas. Al principio del capítulo tienes las instrucciones para instalar los paquetes de otras lenguas. Para ver los que tienes instalados, escribe esto en el

intérprete de comandos interactivo:

```
> > > import pytesseract as tess
```

```
> > > tess.get_languages()
```

```
['afr', 'amh', 'ara', 'asm', 'aze', 'aze_cyrl', 'bel', 'ben', 'bod', 'bos',
```

--fragmento omitido--

```
'ton', 'tur', 'uig', 'ukr', 'urd', 'uzb', 'uzb_cyrl', 'vie', 'yid', 'yor']
```

Las cadenas de esta lista son principalmente códigos de idioma ISO 639-3 de tres caracteres, salvo contadas excepciones. Por ejemplo, mientras que 'aze' es el código ISO 639-3 para el idioma azerí en letras latinas, la cadena 'aze_cyrl' es azerí con caracteres cirílicos. Consulta la documentación de Tesseract para más detalles.

Para escanear imágenes con texto que no está en inglés, pasa uno de estos valores de cadena para el argumento lang. Por ejemplo, frankenstein_jpn.png tiene una traducción al japonés de un fragmento de *Frankenstein*. Descarga este archivo de los recursos en línea del libro e introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > import pytesseract as tess
```

```
> > > from PIL import Image
```

```
> > > img = Image.open('frankenstein_jpn.png')
```

```
> > > text = tess.image_to_string(img, lang='jpn')
```

```
> > > print(text)
```

第 5 話 私が自分の労苦の成果を目の当たりにしたのは、11 月の憂鬱な夜でした。ほとんど苦

痛に等しい不安を抱えながら、私は足元に横たわる生命のないものに存在の輝きを吹き込むこ

--fragmento omitido--

だ有目、しわが寄った顔色、そしてまっすぐな黒い犬と、より恐ろしいコントラストを形成した

だけでした。

Si utilizas el idioma incorrecto, `image_to_string()` devuelve lo que Tesseract ha intentado interpretar en inglés a partir del aspecto de los caracteres japoneses. Obviamente, dado que el texto no está en inglés, lo que devuelve es un galimatías:

```
> > > import pytesseract as tess
> > > from PIL import Image
> > > img = Image.open('frankenstein_jpn.png')
> > > text = tess.image_to_string(img, lang='eng')
> > > print(text)
```

BS FABADOABOMEE AOYEVICLEDIL, 1 ADBBERKCLE, (ELA ER
WISE LW ABBA A TRA B, ALE TIRE DO EMO REVS DICED MS EKA
--*fragmento omitido*--

Para reconocer texto en varias lenguas, puedes combinar los códigos de idioma con un carácter '+' antes de pasárselos al argumento `lang` de la función `image_to_string()`. Por ejemplo, `tess.image_to_string(img, lang='eng+jpn')` reconoce caracteres en inglés y en japonés en una imagen.

La aplicación de escaneo NAPS2

Aunque PyTesseract es útil para extraer texto de imágenes, un caso de uso habitual del OCR es crear documentos PDF a partir de imágenes escaneadas con texto buscable. Existen aplicaciones para hacerlo, pero a menudo no ofrecen la flexibilidad necesaria para automatizar la generación de PDF para cientos o miles de imágenes. Yo recomiendo la aplicación de código abierto Not Another PDF Scanner 2 (NAPS2), no solo por su capacidad para controlar

escáneres planos, sino también por su capacidad para ejecutar Tesseract y añadir texto a documentos PDF. Es gratuita, tiene funciones sencillas y está disponible para Windows, macOS y Linux.

NAPS2 puede combinar varias imágenes en un archivo PDF con texto incrustado sin necesidad de estar conectado a un escáner físico. También sabe cómo utilizar las funciones avanzadas de Tesseract, por lo que puede incrustar las cadenas de texto en su ubicación correcta en las páginas del PDF, y se puede ejecutar desde un *script* de Python.

Instalar y configurar NAPS2

Para instalar NAPS2, ve a <https://www.naps2.com/download> y descarga el instalador para tu sistema operativo. En Linux, descarga el instalador Flatpak para Tesseract. Después, abre una ventana del terminal y ejecuta `flatpak install naps2 -X.X.X-linux-x64.flatpak` (o como se llame el archivo del instalador) desde la carpeta de descargas. Puede que tengas que introducir la contraseña de administrador para completar la instalación.

Una vez instalada, puedes ejecutar la aplicación NAPS2 de escritorio. En Windows, puedes seleccionar NAPS2 en el menú Inicio. En macOS, puedes ejecutar NAPS2 desde Spotlight. En Linux, tendrás que abrir una nueva ventana del terminal y ejecutar `flatpak run com.naps2.Naps2`. No obstante, este libro utiliza NAPS2 desde código Python con el módulo `subprocess` en lugar de la interfaz gráfica de usuario.

Ejecutar NAPS2 desde Python

Los *scripts* de Python pueden utilizar el módulo `subprocess` para ejecutar la aplicación NAPS2 con varios argumentos de línea de comandos. Cuando se ejecuta así, NAPS2 no abre su ventana de aplicación, lo cual es ideal para un paso de automatización en un *script* de Python.

Vamos a usar la imagen `frankenstein.png` de nuevo para hacer que NAPS2 genere un PDF con texto de OCR incrustado. La ubicación del programa NAPS2 es diferente en cada sistema operativo; el

siguiente código muestra la ruta para Windows:

```
> > > import subprocess  
  
> > > naps2_path = [r'C:\Program Files  
NAPS2\NAPS2.Console.exe'] # Windows  
  
> > > proc = subprocess.run(naps2_path + ['-i',  
'frankenstein.png', '-o',  
'output.pdf', '--install', 'ocr-eng', '--ocrlang', 'eng', '-n', '0', '-f',  
'-v'], capture_output=True)
```

En macOS, sustituye la línea que configura la ruta por lo siguiente:
naps2_path = ['/Applications/NAPS2.app/Contents/MacOS/
NAPS2', 'console']. En Linux, utiliza esto: naps2_path = ['flatpak',
'run', 'com.naps2.Naps2', 'console'].

El código crea un nuevo archivo llamado output.pdf que contiene una sola página con la imagen escaneada de frankenstein.png. Sin embargo, si abrimos este archivo con una aplicación para PDF, veremos que se puede seleccionar texto y copiarlo en el portapapeles. Muchas aplicaciones de PDF permitirán guardar el documento como archivo de texto .txt con el texto del OCR.

Veamos cada argumento de línea de comandos de este ejemplo. Puedes cambiarlos como necesites para tus propios fines:

- '-i', 'frankenstein.png': Establece la entrada como la imagen frankenstein.png. Consulta el siguiente apartado con más información sobre cómo especificar múltiples entradas en varios formatos.
- '-o', 'output.pdf': Crea un archivo llamado output.pdf para guardar el resultado del OCR.
- '--install', 'ocr-eng': Instala el paquete de inglés para el OCR. Si el idioma está ya instalado, esto no hace nada. Si quieres instalar un paquete de idioma diferente, utiliza el prefijo ocr- con otro código ISO 639 de tres letras.

•‘--ocrlang’, ‘eng’: Establece el inglés como el idioma que reconoce el escaneo con OCR. Este argumento se pasa directamente al argumento de línea de comandos de Tesseract, así que podrías usar un argumento como ‘eng + jpn + rus’ para especificar que la imagen tiene texto en inglés, japonés y ruso.

•‘-n’, ‘0’: Especifica que queremos hacer cero escaneos y no usamos un escáner plano. Esto evita mensajes de error cuando no hay un escáner conectado al ordenador.

•‘-f’: Obliga a NAPS2 a sobrescribir el archivo de salida output.pdf si ya existe un archivo con ese nombre.

•‘-v’: Habilita el modo verboso para que aparezca texto de estado mientras NAPS2 crea el PDF. Si quieres ver este texto de estado, cambia el argumento `capture_output = True` por `subprocess.run()` por `capture_output = False`.

La documentación en línea para los argumentos de línea de comandos de NAPS2 se encuentra en <https://www.naps2.com/doc/command-line>. En el capítulo 19 vimos el módulo `subprocess` con más detalle.

Especificar entrada

NAPS2 permite importar PDF y la mayoría de los formatos de imagen para crear un PDF final combinado. La aplicación dispone de su propio minilenguaje para especificar varias entradas como un solo argumento de línea de comandos después de `-i`. Esto puede resultar bastante complicado, pero puedes imaginarlo como una delimitación por punto y coma, con índices de Python y notación de rebanadas.

Para especificar varios archivos, sepáralos por punto y coma. Por ejemplo, pasar ‘`-i, ‘cat.png;dog.png;moose.png’`’ crea un PDF con `cat.png` para la primera página, `dog.png` para la segunda página y `moose.png` para la tercera.

También podemos especificar páginas individuales en un PDF con una sintaxis que es idéntica a la de las rebanadas de Python. Pon a continuación del nombre de archivo del PDF el número de página

que quieres usar entre corchetes. Igual que en Python, el 0 representa la primera página. Por ejemplo, pasar `'-i'`, `'spam.pdf[0];spam.pdf[5];eggs.pdf'` crea un PDF con la página 1 de `spam.pdf`, seguida de la página 6 del mismo archivo y todas las páginas de `eggs.pdf`.

Podemos especificar un rango de páginas con esta notación de rebanadas o usar números negativos para representar páginas desde el final del documento PDF. Por ejemplo, pasar `'-i'`, `'spam.pdf[0:2];eggs.pdf[-1]'` combina las dos primeras páginas de `spam.pdf` con la última de `eggs.pdf`.

Hay varias funciones más que ofrece NAPS2 mediante sus argumentos de línea de comandos. Revisa la documentación en línea para descubrirlas. Si ves que NAPS2 no se ajusta a tus necesidades, te recomiendo el paquete `ocrmypdf`, en <https://pypi.org/project/ocrmypdf/> para crear archivos PDF con texto incrustado.

Resumen

En este capítulo, has aprendido cómo aprovechar la potencia de Tesseract para extraer texto de imágenes. Se trata de una función muy potente que puede ahorrarte horas de introducción de datos. Sin embargo, el OCR no hace milagros y es posible que las imágenes necesiten un procesamiento previo para obtener resultados precisos. Tesseract también está diseñado para funcionar con texto oscuro mecanografiado sobre fondos claros en los que el texto está nivelado, y es necesario conocer el idioma del texto de la imagen para obtener buenos resultados.

Los modelos de lenguaje de gran tamaño pueden ayudar a corregir los caracteres reconocidos incorrectamente, pero sus resultados también requieren supervisión humana. Por último, la aplicación de código abierto NAPS2 ofrece una forma de coger varias imágenes y combinarlas en un único PDF con texto OCR incrustado. El OCR es un avance increíble de la informática, pero no es necesario tener un título superior para utilizarlo. Python hace que el OCR sea accesible para todo el mundo.

Preguntas prácticas

1. ¿Qué idioma reconoce Tesseract por defecto?
2. Nombra una biblioteca de imágenes de Python que funcione con PyTesseract.
3. ¿Qué función de PyTesseract acepta un objeto de imagen y devuelve una cadena con el texto de la imagen?
4. Si haces una foto de un cartel en la calle, ¿será Tesseract capaz de identificar el texto del cartel de la foto?
5. ¿Qué función devuelve la lista de paquetes de idiomas instalados para Tesseract?
6. ¿Qué argumento de palabra clave especificamos para PyTesseract si una imagen contiene texto en inglés y en japonés?
7. ¿Qué aplicación permite crear archivos PDF con texto de OCR incrustado?

Programa de práctica: extractor de texto de navegador

Algunos sitios web permiten ver su contenido de texto, pero no dan facilidades para guardarlo o siquiera copiarlo y pegarlo en tu ordenador. Puedes verlos como PDF incrustados dentro de la página web. Tienes un ejemplo de esto en <https://autbor.com/embeddedfrankenstein/>, que se muestra en la figura 22.2.

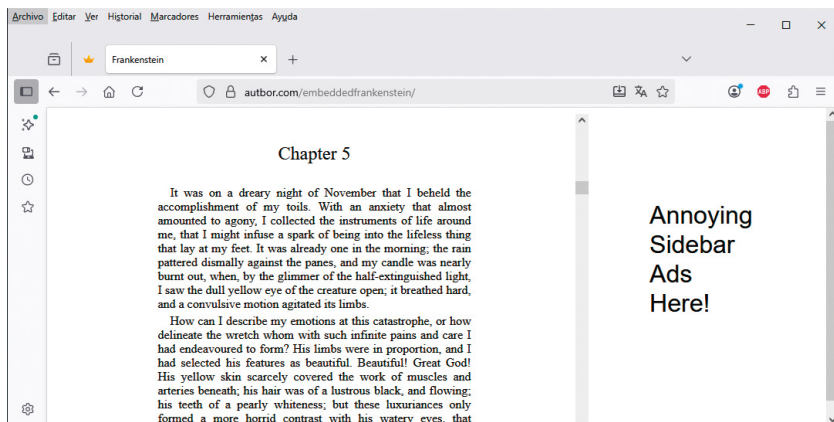


Figura 22.2. Un ejemplo de página web con un documento incrustado.

La biblioteca PyAutoGUI, que veremos en el capítulo 23, puede hacer capturas de pantalla y guardarlas en una imagen, mientras que la biblioteca Pillow, cubierta en el capítulo 21, puede recortar imágenes. PyAutoGUI también cuenta con una aplicación MouseInfo para encontrar coordenadas XY en la pantalla.

Escribe un programa llamado ocrscreen.py que haga una captura de pantalla, recorte la imagen justo a la parte de texto de la captura y se la pase a Py Tesseract para el OCR. El programa debería añadir el texto reconocido al final de un archivo de texto llamado output.txt. Aquí tienes una plantilla para el programa ocrscreen.py:

```
import pyautogui
```

```
# PENDIENTE – Añadir las declaraciones de importación  
adicionales.
```

```
# Las coordenadas de la parte de texto. Ajusta como sea necesario:
```

```
LEFT = 400
```

```
TOP = 200
```

```
RIGHT = 1000
```

```
BOTTOM = 800
```

Hace una captura de pantalla:

```
img = pyautogui.screenshot()
```

Recorta la captura de pantalla a la parte de texto:

```
img = img.crop((LEFT, TOP, RIGHT, BOTTOM))
```

Aplica el OCR a la imagen recortada:

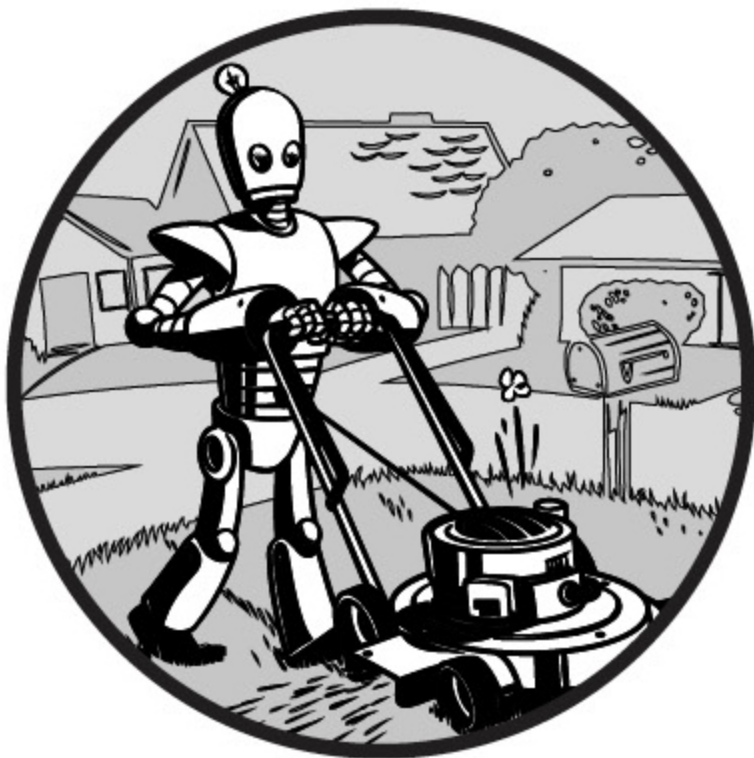
PENDIENTE - Añadir aquí el código de PyTesseract.

Añade el texto del OCR al final de output.txt:

PENDIENTE - Llamar a open() en modo append para añadir el texto del OCR.

Este programa debería permitirte recorrer el texto incrustado que ves en el navegador sin poder guardarlo, ejecutar el programa y mover el PDF a la siguiente página de contenido. Cuando termine, tendrás tu propia copia del texto de los documentos. (Si lees el capítulo 23, también sabrás cómo hacer que tu *script* simule pulsaciones de teclado para moverse solo por la página).

Controlar el ratón y el teclado



Conocer varios paquetes de Python para editar hojas de cálculo, descargar archivos y ejecutar programas es útil, pero a veces simplemente no hay paquetes para las aplicaciones con las que necesitamos trabajar. Las herramientas definitivas para automatizar tareas en el ordenador son los programas que escribimos para controlar directamente el teclado y el ratón. Estos programas pueden enviar a otras aplicaciones pulsaciones de teclas y clics de ratón virtuales, como si estuviéramos sentados frente al ordenador e interactuando con las aplicaciones nosotros mismos.

Esta técnica se conoce como automatización de la interfaz gráfica

de usuario, o automatización de GUI, para abreviar. Con la automatización de GUI, nuestros programas pueden hacer todo lo que puede hacer un usuario humano sentado frente al ordenador, excepto derramar café sobre el teclado. Piensa en la automatización de GUI como en la programación de un brazo robótico. Puedes programar el brazo robótico para que escriba en tu teclado y mueva el ratón por ti. Esta técnica es especialmente útil para tareas que implican muchos clics mecánicos o rellenar formularios. Esta potente técnica es la razón por la que las páginas web de registro e inicio de sesión tienen retos *captcha* para detectar bots. De lo contrario, los programas de automatización podrían registrarse en múltiples cuentas gratuitas, inundar las redes sociales con *spam* o adivinar las contraseñas de las cuentas.

Algunas empresas venden “soluciones de automatización” innovadoras (y caras), que suelen comercializarse como herramientas de automatización de procesos robóticos (RPA). Estos productos no difieren en nada de los *scripts* de Python que puedes crear tú mismo con la biblioteca PyAutoGUI, que tiene funciones para simular movimientos del ratón, clics de botones y pulsaciones de teclas. Este capítulo solo cubre un subconjunto de las características de PyAutoGUI; puedes encontrar la documentación completa en <https://pyautogui.readthedocs.io/>. Para instalar la última versión de PyAutoGUI compatible con este libro, sigue las instrucciones del apéndice A.

Advertencia *No guardes tus programas como pyautogui.py. Si lo haces, al ejecutar `import pyautogui`, Python importará tu programa en vez de PyAutoGUI, y te saldrán mensajes de error como `AttributeError: module 'pyautogui' has no attribute 'click'`.*

Configurar aplicaciones de accesibilidad en macOS

Como medida de seguridad, macOS normalmente no permite que los programas controlen el ratón o el teclado. Para que PyAutoGUI funcione en macOS, debemos configurar el programa que ejecuta nuestro *script* de Python como una aplicación de accesibilidad. Sin este paso, las llamadas a funciones de PyAutoGUI no tendrán ningún efecto.

Tanto si ejecutas tus programas Python desde Mu, IDLE o el terminal, mantén abierta esa aplicación. A continuación, abre Preferencias del sistema y ve a la pestaña Accesibilidad. Las aplicaciones abiertas en ese momento aparecerán bajo la etiqueta Permite que las siguientes aplicaciones controlen el ordenador. Marca Mu, IDLE, Terminal o cualquier otra aplicación que utilices para ejecutar tus *scripts* Python. Se te pedirá que introduzcas tu contraseña para confirmar estos cambios.

Mantener el rumbo

Antes de lanzarte a la automatización de GUI, debes saber cómo evitar los problemas que puedan surgir. Python puede mover el ratón y escribir a una velocidad increíble. De hecho, puede que sea demasiado rápido para que otros programas puedan seguirle el ritmo. Además, si algo sale mal pero el programa sigue moviendo el ratón, será difícil saber exactamente qué está haciendo el programa o cómo solucionar el problema. Igual que las escobas encantadas de la secuencia *El aprendiz de brujo* de la película *Fantasia* de Disney, que seguían llenando (y luego desbordando) el caldero de Mickey con agua, tu programa podría descontrolarse aunque esté siguiendo tus instrucciones al pie de la letra. Detener el programa puede ser difícil si el ratón se mueve por sí solo, impidiéndole hacer clic en la ventana del editor Mu para cerrarla. Por suerte, hay varias formas de prevenir o recuperarse de los problemas de automatización de GUI.

Pausas y medidas de seguridad

Si un programa tiene un error y no podemos utilizar el teclado ni el ratón para cerrarlo, se puede utilizar la función de seguridad de PyAutoGUI. Desliza rápidamente el ratón hacia una de las cuatro esquinas de la pantalla. Cada llamada a una función de PyAutoGUI tiene una pausa de una décima de segundo después de realizar su acción para darnos tiempo suficiente para mover el ratón a una esquina. Si PyAutoGUI detecta que el cursor del ratón está en una esquina, genera la excepción `pyautogui.FailSafeException`. Las instrucciones que no son de PyAutoGUI no tendrán esta pausa. Podemos ajustar la duración de la pausa estableciendo `pyautogui.PAUSE` en un valor distinto de 0.1.

Si te encuentras en una situación en la que necesitas detener tu programa PyAuto GUI, solo tienes que mover rápidamente el ratón hacia una esquina de la pantalla para detenerlo.

Cierre de sesión

Quizás la forma más sencilla de detener un programa de automatización de GUI fuera de control sea cerrar la sesión, lo que cerrará todos los programas en ejecución. En Windows y Linux, la combinación de acceso rápido para cerrar la sesión es **Control-Alt-Supr.** En macOS, es **Comando-Mayús-Q**. Al cerrar la sesión, se perderá cualquier trabajo no guardado, pero al menos no habrá que esperar a que se reinicie el ordenador.

Controlar el movimiento del ratón

En este apartado, aprenderás a mover el ratón y a seguir su posición en la pantalla utilizando PyAutoGUI, pero primero debes comprender cómo funciona PyAutoGUI con las coordenadas.

Las funciones de ratón de PyAutoGUI utilizan coordenadas x e y . La figura 23.1 muestra el sistema de coordenadas de la pantalla del ordenador; es similar al sistema de coordenadas utilizado para las imágenes que vimos en el capítulo 21. El origen, donde x e y son ambos cero, está en la esquina superior izquierda de la pantalla. Las coordenadas x aumentan hacia la derecha y las coordenadas y aumentan hacia abajo. Todas las coordenadas son números enteros positivos; no hay coordenadas negativas.

La resolución es la cantidad de píxeles que tiene la pantalla de alto y de ancho. Si la resolución de la pantalla está configurada en 1.920×1.080 , la coordenada de la esquina superior izquierda será $(0, 0)$, y la de la esquina inferior derecha, $(1919, 1079)$.

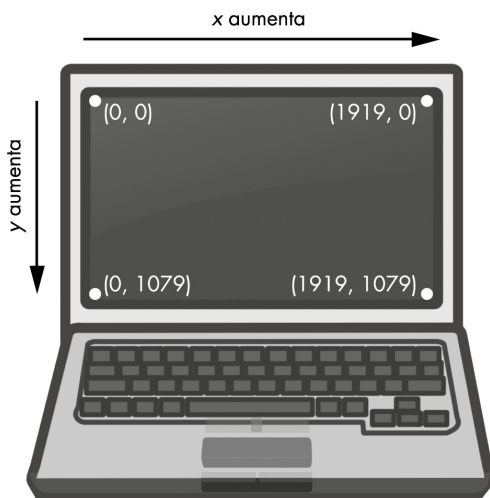


Figura 23.1. Las coordenadas de una pantalla de ordenador con resolución 1.920×1.080 .

La función `pyautogui.size()` devuelve una tupla llamada `Size` con la anchura y la altura de la pantalla en píxeles. Las tuplas con nombre quedan fuera del alcance de este libro, pero básicamente son tuplas con índices enteros que también tienen atributos con nombres. Introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > import pyautogui
```

```
> > > screen_size = pyautogui.size() # Obtiene la resolución de
pantalla.
```

```
> > > screen_size
```

```
Size(width=1920, height=1080)
```

```
> > > screen_size[0], screen_size[1]
```

```
(1920, 1080)
```

```
> > > screen_size.width, screen_size.height
```

```
(1920, 1080)
```

```
> > > tuple(screen_size)
```

(1920, 1080)

La función `pyautogui.size()` devuelve un objeto `Size` de (1920, 1080) en un ordenador con una resolución de 1.920×1.080 ; dependiendo de la resolución de tu pantalla, tu valor de retorno podría ser diferente.

Mover el ratón

Ahora que entiendes las coordenadas de la pantalla, vamos a mover el ratón. La función `pyautogui.moveTo()` moverá instantáneamente el cursor del ratón a una posición específica en la pantalla. Los valores enteros para las coordenadas `x` y `y` constituyen el primer y segundo argumento de la función, respectivamente. Un argumento `duration` opcional, que puede ser entero o flotante, especifica el número de segundos que debe tardar el ratón en desplazarse hasta el destino. Si se omite, el valor predeterminado es 0, lo que implica un movimiento instantáneo. (Todos los argumentos `duration` en las funciones de `PyAutoGUI` son opcionales). Introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > import pyautogui

> > > for i in range(10): # Mueve el ratón en cuadro.

... pyautogui.moveTo(100, 100, duration = 0.25)

... pyautogui.moveTo(200, 100, duration = 0.25)

... pyautogui.moveTo(200, 200, duration = 0.25)

... pyautogui.moveTo(100, 200, duration = 0.25)

...
```

Este ejemplo mueve el ratón en sentido horario formando un cuadrado entre las cuatro coordenadas especificadas un total de 10 veces. Cada movimiento tarda un cuarto de segundo, como especifica el argumento `duration = 0.25`. Si no hubiésemos pasado un tercer argumento a ninguna de las llamadas a `pyautogui.moveTo()`, el cursor del ratón se habría trasladado al

instante de punto a punto.

La función `pyautogui.move()` mueve el cursor en relación con su posición actual. El siguiente ejemplo mueve el ratón en el mismo patrón de cuadro, pero empieza el movimiento desde donde se encuentre el ratón al comenzar la ejecución del código:

```
> > > import pyautogui  
  
> > > for i in range(10):  
  
... pyautogui.move(100, 0, duration = 0.25) # Derecha  
  
... pyautogui.move(0, 100, duration = 0.25) # Abajo  
  
... pyautogui.move(-100, 0, duration = 0.25) # Izquierda  
  
... pyautogui.move(0, -100, duration = 0.25) # Arriba  
  
...
```

La función `pyautogui.move()` también toma tres argumentos: el número de píxeles del movimiento horizontal hacia la derecha, el número de píxeles del movimiento vertical hacia abajo y, opcionalmente, la duración del movimiento. Un entero negativo para el primero o el segundo argumento hará que el ratón se mueva hacia la izquierda o hacia arriba, respectivamente.

Obtener la posición actual

Podemos determinar la posición actual del ratón llamando a la función `pyautogui.position()`, que devolverá una tupla con nombre `Point` de las posiciones `x` y `y` del ratón en el momento de la llamada a la función. Podemos acceder a las coordenadas `x` y `y` mediante los índices enteros 0 y 1 de `Point` o a través de los atributos `x` y `y`. (Esto es similar a los atributos `width` y `height` de la tupla `Size`). Escribe lo siguiente en el intérprete de comandos interactivo, moviendo el ratón después de cada llamada:

```
> > > pyautogui.position() # Obtiene la posición actual del  
ratón.
```

```
Point(x=311, y=622)
```

```
> > > pyautogui.position() # Obtiene la posición actual del ratón de nuevo.
```

```
Point(x=377, y=481)
```

```
> > > p = pyautogui.position() # Y otra vez.
```

```
> > > p
```

```
Point(x=1536, y=637)
```

```
> > > p[0] # La coordenada x está en el índice 0.
```

```
1536
```

```
> > > p.x # La coordenada x también está en el atributo x.
```

```
1536
```

Naturalmente, tus valores de retorno variarán en función de dónde se encuentre tu ratón.

Controlar la interacción del ratón

Ahora que sabes cómo mover el ratón y averiguar dónde está en la pantalla, ya puedes empezar a hacer clics, arrastrar y desplazarte.

Hacer clic

Para enviar un clic virtual al ordenador, llamaremos al método `pyautogui.click()`. Por defecto, este clic usa el botón izquierdo del ratón y se produce donde se encuentre el cursor en ese momento. Si queremos hacer clic en otro sitio, podemos pasar coordenadas `x` e `y` para el clic opcionalmente como primer y segundo argumento.

Si queremos especificar qué botón del ratón hay que usar, incluiremos el argumento `button` con un valor de `'left'`, `'middle'` o `'right'`. Por ejemplo, `pyautogui.click(100, 150, button='left')` hará clic con el botón izquierdo en las coordenadas (100, 150), mientras

que `pyautogui.click(200, 250, button='right')` hará clic con el derecho en (200, 250). Introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > import pyautogui
```

```
> > > pyautogui.click(10, 5) # Mueve el ratón a (10, 5) y hace clic.
```

Deberías ver cómo el puntero del ratón se desplaza hacia la esquina superior izquierda de la pantalla y hace clic una vez. Un “clic” completo se define como pulsar el botón del ratón y soltarlo sin mover el cursor. También puedes hacer clic llamando a `pyautogui.mouseDown()`, que solo pulsa el botón del ratón, y `pyautogui.mouseUp()`, que solo suelta el botón. Estas funciones tienen los mismos argumentos que `click()`; de hecho, `click()` es solo una cómoda envoltura alrededor de estas dos llamadas a funciones.

Para mayor comodidad, la función `pyautogui.doubleClick()` realizará dos clics con el botón izquierdo del ratón. Las funciones `pyautogui.rightClick()` y `pyautogui.middleClick()` realizarán un clic con los botones derecho y central del ratón, respectivamente.

Arrastrar

Arrastrar significa mover el ratón mientras se mantiene pulsado uno de sus botones. Por ejemplo, podemos mover archivos entre carpetas arrastrando sus iconos, igual que podemos mover citas en una aplicación de calendario.

PyAutoGUI cuenta con las funciones `pyautogui.dragTo()` y `pyautogui.drag()` para arrastrar el cursor a una nueva posición o a una posición relativa a la actual. Los argumentos para `dragTo()` y `drag()` son los mismos que para `moveTo()` y `move()`: la coordenada *x*/el movimiento horizontal, la coordenada *y*/el movimiento vertical, y una duración opcional. (El sistema operativo macOS no arrastra bien cuando el ratón se mueve demasiado deprisa, por lo que es aconsejable pasar un argumento `duration`).

Para probar estas funciones, abre una aplicación de diseño gráfico como MS Paint en Windows, Paintbrush en macOS o GNU Paint en

Linux. (Si no tienes aplicaciones de dibujo, puedes usar una en línea en <https://sumopaint.com>). Yo usaré PyAutoGUI para dibujar en estas aplicaciones.

Con el cursor del ratón encima del lienzo de la aplicación de dibujo y la herramienta Lápiz o Pincel seleccionada, escribe lo siguiente en una nueva ventana del editor de archivos y guarda como `spiralDraw.py`:

```
import pyautogui

1 pyautogui.sleep(5)

2 pyautogui.click() # Clic para hacer la ventana activa.

distance = 300

change = 20

while distance > 0:

3 pyautogui.drag(distance, 0, duration=0.2) # Mueve a la derecha.

4 distance = distance - change

5 pyautogui.drag(0, distance, duration=0.2) # Mueve hacia abajo.

6 pyautogui.drag(-distance, 0, duration=0.2) # Mueve a la
izquierda.

distance = distance - change

pyautogui.drag(0, -distance, duration=0.2) # Mueve hacia arriba.
```

Cuando ejecutes este programa, habrá un retardo de cinco segundos 1 durante los cuales puedes mover el cursor sobre la ventana del programa de dibujo con la herramienta Lápiz o Pincel seleccionada. La función `sleep()` de PyAutoGUI es idéntica a `time.sleep()`, pero existe para que no haya que añadir `import time` al código. Después, `spiralDraw.py` tomará el control del ratón y hará clic para activar la ventana del programa de dibujo 2. La ventana activa es la que acepta actualmente la entrada del teclado y las acciones que

realices (como escribir o, en este caso, arrastrar el ratón) afectarán a esa ventana. La ventana activa también se conoce como primer plano. Una vez que el programa de dibujo está activo, `spiralDraw.py` dibuja un patrón de espiral cuadrada como el de la izquierda en la figura 23.2.

La variable `distance` empieza en 300, así que, en la primera iteración del bucle `while`, la primera llamada a `drag()` arrastra el cursor 300 píxeles a la derecha, tardando 0,2 segundos ³. Después, `distance` se reduce a 280 ⁴, y la segunda llamada a `drag()` arrastra el cursor 280 píxeles hacia abajo ⁵. La tercera llamada a `drag()` arrastra el cursor -280 horizontalmente (280 hacia la izquierda) ⁶, `distance` se reduce a 260, y la última llamada a `drag()` arrastra el cursor 260 píxeles hacia arriba. En cada iteración, el ratón se arrastra a la derecha, hacia abajo, hacia la izquierda y hacia arriba, y `distance` es ligeramente menor que en la iteración anterior. Pasando en bucle por este código, podemos mover el cursor para dibujar una espiral cuadrada.

Aunque también podamos crear una imagen de espiral cuadrada con el paquete `Pillow`, explicado en el capítulo 21, crear la imagen controlando el ratón para dibujar en MS Paint nos permite usar los distintos estilos de pincel del programa, como muestra la figura 23.2, a la derecha, además de otras funciones avanzadas, como los degradados y el relleno. Puedes preseleccionar el pincel (o hacer que el código seleccione esos ajustes) y después ejecutar el programa para dibujar espirales.

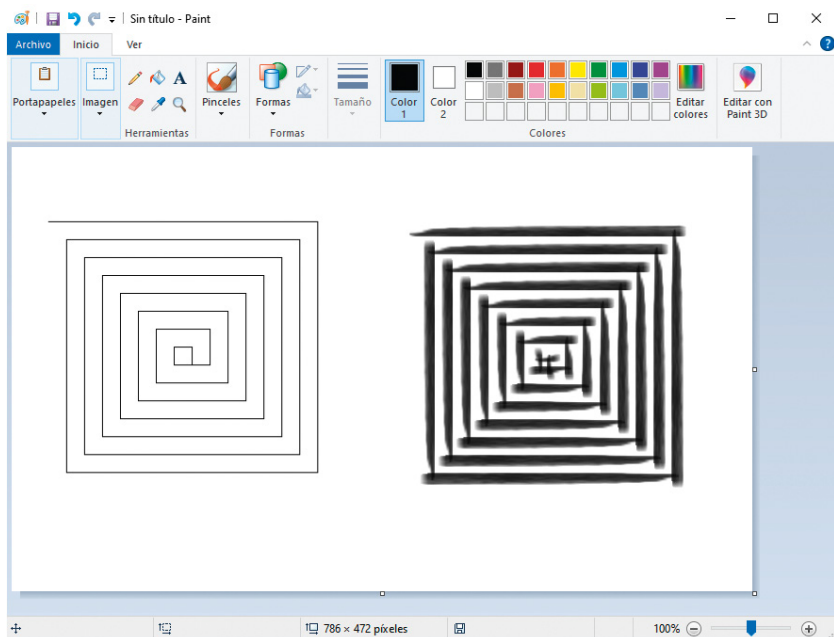


Figura 23.2. Los resultados del ejemplo `pyautogui.drag()`, dibujados con distintos pinceles de MS Paint.

Podríamos dibujar esta espiral a mano (o mejor dicho, a ratón), pero tendríamos que trabajar despacio para no perder precisión. ¡PyAutoGUI puede hacerlo en segundos!

Desplazarse

La última función del ratón de PyAutoGUI es `scroll()`, a la que se pasa un argumento entero que indica cuántas unidades se desea desplazar el ratón hacia arriba o hacia abajo. El tamaño de una unidad varía dependiendo del sistema operativo y la aplicación, por lo que tendrás que probar para ver exactamente cuánto se desplaza en tu situación particular. El desplazamiento se realiza en la posición actual del cursor del ratón. Si se pasa un entero positivo, se desplaza hacia arriba y, si se pasa un entero negativo, se desplaza hacia abajo. Ejecuta lo siguiente en el intérprete de comandos interactivo del editor Mu mientras el cursor del ratón se encuentra sobre la ventana del editor:

> > > **pyautogui.scroll(200)**

Verás que Mu sube si el cursor del ratón está sobre un campo de texto por el que se pueda subir.

Planificar los movimientos del ratón

Una de las dificultades de escribir un programa que automatice los clics en la pantalla es dar con las coordenadas *x* e *y* de las cosas en las que hay que hacer clic. La función `pyautogui.mouseInfo()` puede ayudarnos con esto.

La función `pyautogui.mouseInfo()` está pensada para llamarse desde el intérprete de comandos interactivo, más que como parte de un programa. Inicia una aplicación llamada `MouseInfo` que se incluye en `PyAutoGUI`. La figura 23.3 muestra la ventana de la aplicación.

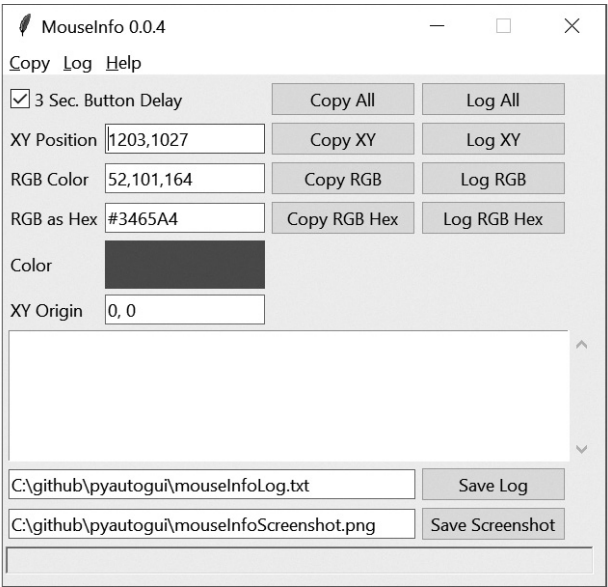


Figura 23.3. La ventana de la aplicación `MouseInfo`.

Introduce lo siguiente en el intérprete de comandos interactivo:

> > > **import pyautogui**

> > > `pyautogui.mouseInfo()`

Esto hace que aparezca la ventana de `MouseInfo`, que proporciona información sobre la posición actual del cursor, así como el color del píxel situado debajo del cursor, en forma de tupla RGB de tres enteros y como valor hexadecimal. El color en sí aparece en el cuadro de color de la ventana.

Para ayudarnos a registrar esta información de coordenadas o píxeles, podemos hacer clic en uno de los ocho botones para copiar o registrar. Los botones `Copy All`, `Copiar XY`, `Copy RGB` y `Copy RGB Hex` copiarán su información respectiva al portapapeles. Los botones `Log All`, `Log XY`, `Log RGB` y `Log RGB Hex` escribirán la información correspondiente en el campo de texto grande de la ventana. Podemos guardar el texto de este campo de texto de registro haciendo clic en el botón `Save Log`.

Por defecto, la casilla de verificación `3 Sec. Button Delay` está seleccionada, lo que provoca un retraso de tres segundos entre el clic en un botón y la copia o el registro correspondiente. Esto nos da un breve periodo de tiempo para hacer clic en el botón y luego mover el ratón a la posición deseada. Puede resultar más fácil deseleccionar esta casilla, mover el ratón a la posición deseada y pulsar las teclas **F1** a **F8** para copiar o registrar la posición del ratón. Puedes consultar los menús de copia y registro en la parte superior de la ventana `MouseInfo` para averiguar qué tecla se asigna a cada botón.

Por ejemplo, deselecciona `3 Sec. Button Delay`, mueve el ratón por la pantalla mientras pulsas **F6** y observa cómo se registran las coordenadas `x` e `y` del ratón en el campo de texto grande en el centro de la ventana. Más tarde podrás utilizar estas coordenadas en *scripts* de `PyAutoGUI`.

Para más información sobre `MouseInfo`, repasa la documentación complementaria en <https://mouseinfo.readthedocs.io/>.

Hacer capturas de pantalla

Los programas de automatización de GUI no tienen por qué hacer clic o escribir a ciegas. `PyAutoGUI` cuenta con funciones de captura

de pantalla que pueden crear un archivo de imagen basado en el contenido actual de la pantalla. Estas funciones también pueden devolver un objeto Image de Pillow del aspecto de la pantalla actual. Si no has leído el libro entero hasta aquí, te conviene leer el capítulo 21 e instalar el paquete Pillow antes de seguir con este apartado.

Para hacer capturas de pantalla en Python, llama a la función `pyautogui.screenshot()`. Escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > import pyautogui
```

```
> > > im = pyautogui.screenshot()
```

La variable `im` contendrá el objeto Image de la captura. Ahora puedes llamar a métodos con el objeto Image en la variable `im`, igual que con cualquier otro objeto Image. El capítulo 21 ofrece más información sobre estos objetos.

Supongamos que uno de los pasos de tu programa de automatización de GUI consiste en hacer clic en un botón gris. Antes de llamar al método `click()`, puede que te convenga hacer una captura de pantalla y observar el píxel en el que el *script* está a punto de hacer clic. Si no es del mismo color gris que el botón gris, el programa sabrá que algo va mal. Quizás la ventana se haya movido inesperadamente, o tal vez un cuadro de diálogo emergente haya bloqueado el botón. En este punto, en lugar de continuar y posiblemente causar estragos al hacer clic en el elemento equivocado, el programa puede detenerse por sí mismo.

Puedes obtener el valor de color RGB de un píxel concreto de la pantalla con la función `pixel()`. Escribe esto en el intérprete de comandos interactivo:

```
> > > import pyautogui
```

```
> > > pyautogui.pixel(0, 0)
```

```
(176, 176, 175)
```

```
> > > pyautogui.pixel((50, 200))
```

(130, 135, 144)

Pasa a `pixel()` dos enteros para una coordenada XY y te dirá el color del píxel que ocupa esas coordenadas en tu imagen. El valor de retorno de `pixel()` es una tupla RGB de tres enteros para la cantidad de rojo, verde y azul del píxel. (No hay un cuarto valor para alfa porque las capturas de pantalla son imágenes totalmente opacas).

La función `pixelMatchesColor()` de PyAutoGUI devolverá True si el píxel en la posición x e y coincide con ese color. El primer y el segundo argumento son enteros para las coordenadas x e y, y el tercero es una tupla de tres enteros para el color RGB con el que debe coincidir el píxel. Introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > import pyautogui
```

```
1 > > > pyautogui.pixel((50, 200))
```

(130, 135, 144)

```
2 > > > pyautogui.pixelMatchesColor(50, 200, (130, 135, 144))
```

True

```
3 > > > pyautogui.pixelMatchesColor(50, 200, (255, 135, 144))
```

False

Después de usar `pixel()` para obtener una tupla RGB para el color de un píxel en unas coordenadas determinadas 1, pasa esas mismas coordenadas y tupla RGB a `pixelMatchesColor()` 2, que debería devolver True. A continuación, cambia un valor de la tupla RGB y vuelve a llamar a `pixelMatchesColor()` con las mismas coordenadas 3. El valor de retorno debería ser False. Puede ser útil llamar a este método siempre que tus programas de automatización de GUI estén a punto de llamar a `click()`. Observa que el color en las coordenadas dadas debe coincidir a la perfección. Si es ligeramente diferente, por ejemplo (255, 255, 254) en vez de (255, 255, 255), `pixelMatchesColor()` devolverá False.

Reconocimiento de imágenes

Pero ¿y si no sabes de antemano dónde debe hacer clic PyAutoGUI? En ese caso, puedes utilizar el reconocimiento de imágenes. Da a PyAutoGUI una imagen de lo que deseas seleccionar y deja que calcule las coordenadas. Por ejemplo, si previamente has hecho una captura de pantalla para obtener la imagen de un botón Enviar en submit.png, la función locateOnScreen() devolverá las coordenadas donde se encuentra esa imagen. Para ver cómo funciona locateOnScreen(), prueba a hacer una captura de una pequeña área de tu pantalla; a continuación, guarda la imagen e introduce lo siguiente en el intérprete de comandos interactivo, sustituyendo submit.png por el nombre de archivo de tu captura de pantalla:

```
> > > import pyautogui
```

```
> > > box = pyautogui.locateOnScreen('submit.png')
```

```
> > > box
```

```
Box(left = 643, top = 745, width = 70, height = 29)
```

```
> > > box[0]
```

```
643
```

```
> > > box.left
```

```
643
```

El objeto Box es una tupla que devuelve locateOnScreen() y tiene la coordenada x del borde izquierdo, la coordenada y del borde superior, la anchura y la altura para el primer lugar de la pantalla en el que se ha encontrado la imagen. Si estás probando esto en tu ordenador con tu propia captura de pantalla, tu valor de retorno será distinto del que se muestra aquí.

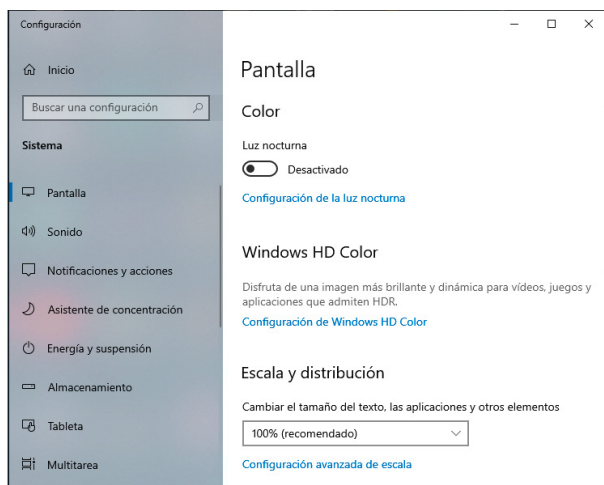
Si la imagen no se encuentra en la pantalla, locateOnScreen() lanza una excepción ImageNotFoundException. Ten en cuenta que la imagen de la pantalla debe coincidir a la perfección con la imagen suministrada para que pueda reconocerse. Si la imagen no coincide aunque sea solo en un píxel, locateOnScreen() lanza la excepción ImageNotFoundException. Si has cambiado la resolución de tu pantalla, las imágenes de capturas anteriores podrían no coincidir

con las de tu pantalla actual porque tienen un factor de escala diferente. Los factores de escala están fuera del alcance de este libro, pero a veces se usan en monitores modernos de alta resolución. Puedes cambiar la escala en los ajustes de pantalla de tu sistema operativo, como se muestra en la figura 23.4.

Si la imagen se encuentra en varios lugares de la pantalla, `locateAllOnScreen()` devolverá un objeto `Generator`. Estos objetos están fuera del ámbito de este libro, pero podemos pasarlos a `list()` para que devuelva una lista de objetos `Box`. Habrá un objeto `Box` por cada ubicación donde se encuentre la imagen en la pantalla. Continúa con el ejemplo del intérprete de comandos interactivo introduciendo lo siguiente (cambiando `'submit.png'` por el nombre de archivo de tu imagen):

```
> > > list(pyautogui.locateAllOnScreen('submit.png'))
```

```
[(643, 745, 70, 29), (1007, 801, 70, 29)]
```



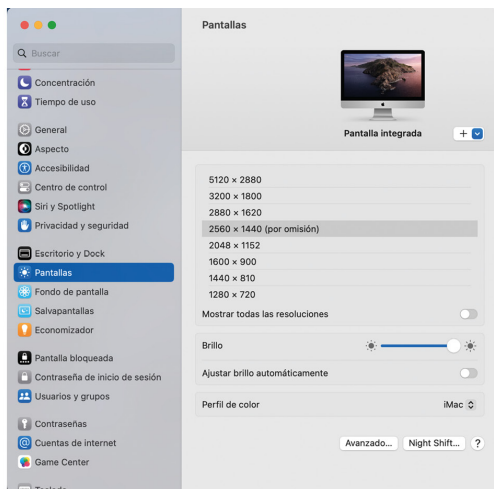


Figura 23.4. Los ajustes de escala y resolución en Windows y macOS.

En este ejemplo, la imagen aparece en dos sitios. Si solo apareciese en uno, llamar a `list(locateAllOnScreen())` devolvería una lista con un solo objeto `Box`.

Una vez que tenemos el objeto `Box` para la imagen concreta que queremos seleccionar, podemos hacer clic en el centro de esa área pasando la tupla a `click()`. Escribe esto en el intérprete de comandos interactivo:

```
> > > pyautogui.click((643, 745, 70, 29))
```

Como atajo, también se puede pasar directamente el nombre de archivo de la imagen a la función `click()`:

```
> > > pyautogui.click('submit.png')
```

Las funciones `moveTo()` y `dragTo()` también aceptan argumentos de nombre de imagen. Recuerda que `locateOnScreen()` lanza una excepción si no puede encontrar la imagen en la pantalla, así que deberías llamarla desde una declaración `try`:

`try`:

```
location = pyautogui.locateOnScreen('submit.png')
```

`except pyautogui.ImageNotFoundException:`

`print("Image could not be found.")`

Sin las declaraciones `try` y `except`, la excepción no manejada detendría el programa. Dado que no podemos estar seguros de que nuestro programa encontrará siempre la imagen, es buena idea usar declaraciones `try` y `except` al llamar a `locateOnScreen()`. En versiones de PyAutoGUI anteriores a 1.0.0, `locateOnScreen()` devolvía `None` en vez de lanzar una excepción. Podemos llamar a `pyautogui.useImageNotFoundException()` en versiones antiguas para lanzar una excepción, o a `pyautogui.useImageNotFoundException(False)` en versiones modernas para devolver `None`.

Obtener información de la ventana

El reconocimiento de imágenes es una forma delicada de encontrar cosas en la pantalla; solo con que haya un píxel de un color diferente, `pyautogui.locateOnScreen()` no encontrará la imagen. Si necesitas encontrar una ventana en concreto en una pantalla, es más fácil y fiable usar las funciones de ventana de PyAutoGUI.

Nota *En la versión 1.0.0, las funciones de ventana de PyAutoGUI solo funcionan en Windows, no en macOS ni en Linux. Estas funciones se derivan de la inclusión del paquete PyGetWindow en PyAutoGUI.*

Obtener la ventana activa

La ventana activa en la pantalla es la ventana que se encuentra actualmente en primer plano y que acepta entrada del teclado. Si estás ahora mismo escribiendo código en Mu, la ventana del editor es la ventana activa. De todas las ventanas de la pantalla, solo una estará activa a la vez. En el intérprete de comandos interactivo, llama a la función `pyautogui.getActiveWindow()` para obtener un objeto `Window` (técnicamente un objeto `Win32Window` cuando se ejecuta en Windows). Una vez que tengas ese objeto, puedes recuperar cualquiera de sus atributos, que describen su tamaño, posición y título:

- left, right, top, bottom: Un único entero para la coordenada x o y del lado de la ventana.

- topleft, topright, bottomleft, bottomright: Una tupla Point de dos enteros para las coordenadas (x, y) de la esquina de la ventana.

- midleft, midright, midtop, midbottom: Una tupla Point de dos enteros para las coordenadas (x, y) del centro de los lados de la ventana.

- width, height: Un único entero para una de las dimensiones de la ventana, en píxeles.

- size: Una tupla Size de dos enteros para la (anchura, altura) de la ventana.

- area: Un solo entero que representa el área de la ventana, en píxeles.

- center: Una tupla Point de dos enteros para las coordenadas (x, y) del centro de la ventana.

- centerx, centery: Un solo entero para la coordenada x o y del centro de la ventana.

- box: Una tupla de cuatro números enteros para las medidas (izquierda, arriba, ancho, alto) de la ventana.

- title: Una cadena con el texto de la barra de título en la parte superior de la ventana.

Para obtener la posición, el tamaño y el título de la ventana a partir del objeto window, por ejemplo, introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > import pyautogui
```

```
> > > active_win = pyautogui.getActiveWindow()
```

```
> > > active_win
```

```
Win32Window(hwnd = 2034368)
```

```
> > > str(active_win)
```

```
'<Win32Window left="500", top="300", width="2070",  
height="1208", title="Mu 1.0.1 – test1.py">'
```

```
> > > active_win.title
```

```
'Mu 1.0.1 – test1.py'
```

```
> > > active_win.size
```

```
Size(width=2070, height=1208)
```

```
> > > active_win.left, active_win.top, active_win.right,  
active_win.bottom
```

```
(500, 300, 2570, 1508)
```

```
> > > active_win.topleft
```

```
Point(x=500, y=300)
```

```
> > > pyautogui.click(active_win.left + 10, active_win.top +  
20)
```

Ahora puedes utilizar estos atributos para calcular coordenadas precisas dentro de una ventana. Si sabes que el botón en el que quieres hacer clic se encuentra siempre 10 píxeles a la derecha y 20 píxeles por debajo de la esquina superior izquierda de la ventana, y que la esquina superior izquierda de la ventana se encuentra en las coordenadas de pantalla (300, 500), llamar a `pyautogui.click(310, 520)` (o `pyautogui.click(active_win.left + 10, active_win.top + 20)` si `active_win` contiene el objeto `Window` para la ventana) hará clic en el botón. De ese modo, no tendrás que depender de la función `locateOnScreen()`, más lenta y menos fiable, para encontrar el botón.

Buscar ventanas con otras funciones

Aunque `getActiveWindow()` es útil para obtener la ventana que está activa en el momento de la llamada a la función, necesitaremos utilizar alguna otra función para obtener objetos `Window` para las

demás ventanas de la pantalla. Las tres funciones siguientes devuelven una lista de objetos Window. Si no encuentran ninguna ventana, devuelven una lista vacía:

- `pyautogui.getAllWindows()`: Devuelve una lista de objetos Window para cada ventana visible en la pantalla.

- `pyautogui.getWindowsAt(x, y)`: Devuelve una lista de objetos Window para cada ventana visible que incluye el punto (x, y).

- `pyautogui.getWindowsWithTitle(title)`: Devuelve una lista de objetos Window para cada ventana visible que incluya la cadena title en la barra de título.

PyAutoGUI también cuenta con una función `pyautogui.getAllTitles()`, que devuelve una lista de cadenas de todas las ventanas visibles.

Manipular ventanas

Los atributos de ventana tienen más utilidades que simplemente indicar el tamaño y la posición de la ventana. También podemos configurar sus valores para redimensionar o mover la ventana. Por ejemplo, escribe esto en el intérprete de comandos interactivo:

```
> > > import pyautogui
```

```
> > > active_win = pyautogui.getActiveWindow()
```

```
1 > > > active_win.width # Obtiene la anchura actual de la  
ventana.
```

```
1669
```

```
2 > > > active_win.topleft # Obtiene la posición actual de la  
ventana.
```

```
Point(x=174, y=153)
```

```
3 > > > active_win.width = 1000 # Cambia la anchura.
```

```
4 > > > active_win.topleft = (800, 400) # Mueve la ventana.
```

Primero, usamos los atributos del objeto Window para averiguar información sobre el tamaño 1 y la posición 2 de la ventana. Después de llamar a estas funciones en el editor Mu, la ventana debería hacerse más estrecha 3 y moverse 4, como en la figura 23.5.

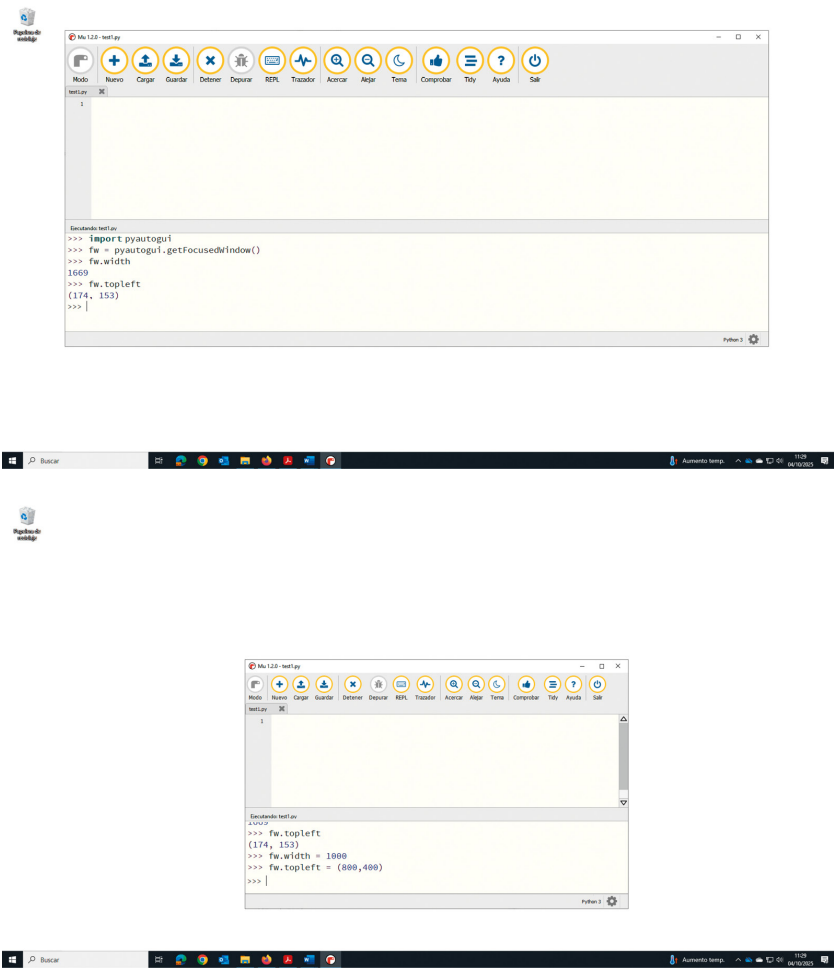


Figura 23.5. La ventana del editor Mu antes (arriba) y después (abajo) de usar los atributos del objeto Window para redimensionarla y moverla.

También se puede buscar y cambiar los estados minimizada, maximizada y activa de la ventana. Prueba a escribir lo siguiente en el intérprete de comandos interactivo:

```
> > > import pyautogui
```

```
> > > active_win = pyautogui.getActiveWindow()
```

```
> > > active_win.isMaximized # Devuelve True si la ventana está maximizada
```

False

```
> > > active_win.isMinimized # Devuelve True si la ventana está minimizada
```

False

```
> > > active_win.isActive # Devuelve True si la ventana es la ventana activa
```

True

```
> > > active_win.maximize() # Maximiza la ventana
```

```
> > > active_win.isMaximized
```

True

```
> > > active_win.restore() # Deshace una acción de minimizar/ maximizar
```

```
> > > active_win.minimize() # Minimiza la ventana
```

```
> > > import time
```

```
> > > # Espera 5 segundos mientras activamos otra ventana:
```

```
> > > time.sleep(5); active_win.activate()
```

```
> > > active_win.close() # Esto cierra la ventana en la que estamos escribiendo.
```

Los atributos `isMaximized`, `isMinimized` e `isActive` contienen valores booleanos que indican si la ventana se encuentra actualmente en ese estado. Los métodos `maximize()`, `minimize()`, `activate()` y `restore()` cambian el estado de la ventana. Después de maximizar o

minimizar la ventana con `maximize()` o `minimize()`, el método `restore()` restaurará la ventana a su tamaño y posición anteriores.

El método `close()` cerrará una ventana. Ten cuidado con este método porque puede omitir los cuadros de diálogo que preguntan si quieres guardar el trabajo antes de salir de la aplicación.

Consulta la documentación de PyAutoGUI para ver todos los detalles sobre sus funciones para controlar ventanas.

Captchas y ética informática

Las pruebas de Turing completamente automáticas y públicas para diferenciar a las máquinas de los humanos (CAPTCHA, *Completely Automated Public Turing test to tell Computers and Humans Apart*) son pequeñas pruebas que nos piden escribir las letras de una imagen distorsionada o hacer clic en las fotos con una boca de agua, por ejemplo. Son pruebas fáciles, aunque molestas, para los humanos, pero casi imposibles de resolver para un software. Después de leer este capítulo, verás lo fácil que es escribir un *script* que pueda, por ejemplo, registrarse para miles de millones de cuentas de correo electrónico e invadir a los usuarios con mensajes hostigadores. Los *captchas* alivian esto requiriendo un paso que solo un humano pueda superar.

Pero no todos los sitios web implementan *captchas*, por lo que pueden ser vulnerables al abuso de programadores poco éticos. Aprender a programar es una habilidad potente y emocionante y puedes sentir la tentación de usar mal este poder para una ganancia personal o para presumir. Pero, igual que una puerta que no está cerrada con llave no es una invitación a entrar, la responsabilidad de tus programas es tuya como programador. No es buena idea buscar la vuelta a los sistemas para causar daños, invadir la privacidad u obtener una ventaja ilegítima. Espero que mi esfuerzo para escribir este libro te permita ser una persona más productiva, no un mercenario.

Controlar el teclado

PyAutoGUI también tiene funciones para enviar pulsaciones de teclado virtuales a tu ordenador, lo que permite rellenar formularios

o introducir texto en aplicaciones.

Enviar cadenas de pulsaciones de teclas

La función `pyautogui.write()` envía pulsaciones de teclas virtuales al ordenador. Lo que hagan estas pulsaciones dependerá de qué ventana esté activa y qué campo de texto tenga el foco. A veces, conviene enviar primero un clic de ratón al campo de texto que queremos para asegurarnos de que el foco está ahí.

Para ver un sencillo ejemplo, vamos a usar Python para escribir automáticamente las palabras `Hello, world!` en una ventana del editor. Primero, abre una nueva ventana en el editor de archivos y colócala en la esquina superior izquierda de la pantalla para que PyAutoGUI haga clic en el lugar correcto para ponerle el foco. A continuación, escribe esto en el intérprete de comandos interactivo:

```
> > > pyautogui.click(100, 200); pyautogui.write('Hello,  
world!')
```

Observa cómo poner dos comandos en la misma línea, separados por punto y coma, evita que el intérprete de comandos interactivo te pida que introduzcas algo entre la ejecución de las dos instrucciones. Así evitamos poner el foco por accidente en otra ventana entre las llamadas a `click()` y a `write()`, lo cual estropearía el ejemplo.

Python enviará primero un clic virtual a las coordenadas (100, 200), lo que debería ser la ventana del editor de archivos para poner el foco en ella. La llamada a `write()` enviará el texto `Hello, world!` a la ventana, produciendo un resultado como la figura 23.6. ¡Ya tienes código que pueda escribir por ti!

Por defecto, la función `write()` introducirá la cadena entera al instante. Sin embargo, podemos pasar un segundo argumento opcional para añadir una breve pausa entre caracteres. Este segundo argumento es un valor entero o flotante del número de segundos de la pausa. Por ejemplo, `pyautogui.write('Hello, world!', 0.25)` esperará un cuarto de segundo después de escribir la `H`, otro cuarto de segundo después de escribir la `e`, y así sucesivamente. Este efecto de escritura gradual puede ser útil para aplicaciones más lentas que

no pueden procesar las pulsaciones de teclado lo bastante rápido como para ir al ritmo de PyAutoGUI.

Para caracteres como A o !, PyAutoGUI simulará automáticamente la pulsación de la tecla **Mayús**.

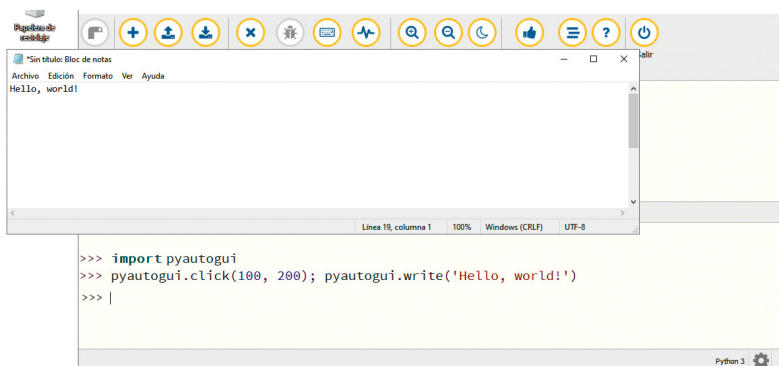


Figura 23.6. Uso de PyAutoGUI para hacer clic en la ventana del editor de archivos y escribir “Hello, world!”.

Especificar nombres de teclas

No todas las teclas son fáciles de representar con caracteres de texto únicos. Por ejemplo, ¿cómo se representa la tecla **Mayús** o la tecla de la flecha izquierda como un solo carácter? En PyAutoGUI, estas teclas se representan mediante valores de cadena cortos: ‘esc’ para la tecla **Esc** o ‘enter’ para **Intro**.

En lugar de un único argumento de cadena, se puede pasar una lista de estas cadenas de teclas a write(). Por ejemplo, la siguiente llamada pulsa **A**, luego **B**, luego la flecha izquierda dos veces (moviendo el cursor delante de la “a”) y, por último, las teclas **X** e **Y**:

```
> > > pyautogui.click(100, 200); pyautogui.write('Hello, world!')
```

Como pulsar la tecla de la flecha izquierda mueve el cursor del teclado, el resultado será XYab. La tabla 23.1 enumera las cadenas de teclas del teclado PyAutoGUI que podemos pasar a write() para

simular la pulsación de cualquier combinación de teclas.

También puedes consultar la lista `pyautogui.KEYBOARD_KEYS` para ver todas las cadenas de teclas posibles que aceptará PyAutoGUI. La cadena 'shift' se refiere a la tecla **Mayús** izquierda y equivale a 'shiftright'. Lo mismo se aplica a las cadenas 'ctrl', 'alt' y 'win'; todas ellas se refieren a la tecla del lado izquierdo.

Tabla 23.1. Valores de cadena para las teclas del teclado.

Cadena	Significado de la tecla
<code>space</code>	La tecla 'Esp' o 'Barra', caracteres '@', '#', etc.
<code>enter</code>	La tecla 'Enter' o '\n')
<code>esc</code>	La tecla Esc.
<code>shiftleft</code>	La tecla Mayús izquierda.
<code>shiftright</code>	La tecla Mayús derecha.
<code>ctrlleft</code>	La tecla Ctrl izquierda.
<code>ctrlright</code>	La tecla Ctrl derecha.
<code>tab</code>	La tecla Tab.
<code>backspace</code>	La tecla Borrar o Supr.
<code>pageup</code>	La tecla Pág. Arriba.
<code>pagedown</code>	La tecla Pág. Abajo.
<code>home</code>	La tecla Inicio.
<code>end</code>	La tecla Fin.
<code>up</code>	La tecla Flecha hacia arriba.
<code>down</code>	La tecla Flecha hacia abajo.
<code>left</code>	La tecla Flecha hacia la izquierda.
<code>right</code>	La tecla Flecha hacia la derecha.
<code>volumeup</code>	La tecla para aumentar el volumen.
<code>volumedown</code>	La tecla para bajar el volumen.
<code>pause</code>	La tecla Pausa.
<code>capslock</code>	La tecla Bloq. Mayús.
<code>numlock</code>	La tecla Bloq. Num y Bloq. Despl.
<code>insert</code>	La tecla Insert.
<code>printscreen</code>	La tecla Pant.
<code>winleft</code>	La tecla Win izquierda (en Windows).
<code>winright</code>	La tecla Win derecha (en Windows).
<code>command</code>	La tecla Comando (en macOS).
<code>option</code>	La tecla Opción (en macOS).

Pulsar y soltar teclas

Al igual que las funciones `mouseDown()` y `mouseUp()`, `pyautogui.keyDown()` y `pyautogui.keyUp()` envían pulsaciones y liberaciones de teclas virtuales al ordenador. Se les pasa una cadena de teclas (véase la tabla 23.1) como argumento. Para mayor

comodidad, PyAutoGUI proporciona la función `pyautogui.press()`, que llama a ambas funciones para simular una pulsación completa de tecla.

Ejecuta el siguiente código, que escribirá el carácter del signo de dólar (\$) (que se obtiene manteniendo pulsada la tecla **Mayús** y mientras se pulsa **4**):

```
> > > pyautogui.keyDown('shift'); pyautogui.press('4');  
pyautogui.keyUp('shift')
```

Esta línea pulsa y mantiene pulsada **Mayús**, pulsa y suelta **4** y luego suelta **Mayús**. Si necesitas escribir una cadena en un campo de texto, la función `write()` es la más adecuada, pero, para aplicaciones que requieren comandos de una sola tecla, la función `press()` es la opción más sencilla.

Ejecutar combinaciones de teclas rápidas

Una “tecla rápida” o “atajo” es una combinación de teclas que se pulsa para invocar alguna función de la aplicación. El atajo habitual para copiar una selección es **Control-C** (en Windows y Linux) o **Comando-C** (en macOS). El usuario mantiene pulsada la tecla **Control**, pulsa la tecla **C** y, a continuación, suelta las teclas **C** y **Control**.

Para hacer esto con las funciones `keyDown()` y `keyUp()` de PyAutoGUI, tendríamos que escribir lo siguiente:

```
pyautogui.keyDown('ctrl')
```

```
pyautogui.keyDown('c')
```

```
pyautogui.keyUp('c')
```

```
pyautogui.keyUp('ctrl')
```

Esto es bastante complicado. En su lugar, vamos a usar la función `pyautogui.hotkey()`, que toma varios argumentos de cadenas de teclas, las pulsa en orden y las suelta en orden inverso. Por ejemplo, para el ejemplo de **Control-C**, el código sería simplemente este:

`pyautogui.hotkey('ctrl', 'c')`

Esta función es especialmente útil para atajos grandes. En Word, la combinación **Control-Alt-Mayús-S** muestra el panel Estilos. En lugar de hacer ocho llamadas a funciones (cuatro a `keyDown()` y cuatro a `keyUp()`), podemos sencillamente llamar a `hotkey('ctrl', 'alt', 'shift', 's')`.

Configurar scripts de automatización de GUI

Los *scripts* de automatización de GUI son una forma estupenda de automatizar las tareas aburridas, pero también pueden ser muy delicados. Si una ventana está en el lugar equivocado en el Escritorio o aparece una ventana emergente inesperadamente, el *script* podría hacer clic en los elementos equivocados de la pantalla. Aquí tienes algunos consejos para configurar tus *scripts* de automatización de GUI:

- Utiliza la misma resolución de pantalla cada vez que ejecutes el *script* para que la posición de las ventanas no cambie.
- La ventana de la aplicación en la que hace clic el *script* debe estar maximizada para que sus botones y menús estén en el mismo lugar cada vez que ejecutes el *script*.
- Añade pausas generosas mientras se carga el contenido; no conviene que el *script* empiece a hacer clic antes de que la aplicación esté lista.
- Utiliza `locateOnScreen()` para encontrar los botones y menús en los que hacer clic, en lugar de basarte en coordenadas. Si el *script* no encuentra lo que necesita para hacer clic, detén el programa en lugar de dejar que continúe haciendo clic a ciegas.
- Utiliza `getWindowsWithTitle()` para asegurarte de que la ventana de la aplicación en la que crees que está haciendo clic el *script* existe, y utiliza el método `activate()` para poner esa ventana en primer plano.
- Utiliza el módulo logging del capítulo 5 para mantener un archivo de registro de lo que ha hecho el *script*. De esta manera, si tienes

que detenerlo a mitad de un proceso, podrás cambiarlo para que continúe desde donde se quedó.

- Añade tantas comprobaciones como puedas a tu *script*. Piensa en cómo podría fallar si aparece una ventana emergente inesperada o si el ordenador pierde la conexión a Internet.

- Puede que te interese supervisar el *script* cuando se inicie por primera vez para asegurarte de que funciona correctamente.

También podría ser interesante poner una pausa al inicio del *script* para que el usuario pueda configurar la ventana en la que el *script* hará clic. PyAutoGUI tiene una función `sleep()` que actúa de forma idéntica a `time.sleep()` (pero nos libera de tener que añadir `import time` a nuestros *scripts*). También hay una función `countdown()` que imprime números en cuenta atrás para dar al usuario una indicación visual de que el *script* continuará pronto. Introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > import pyautogui
```

```
> > > pyautogui.sleep(3) # Pausa el programa 3 segundos.
```

```
> > > pyautogui.countdown(10) # Cuenta atrás de 10 segundos.
```

```
10 9 8 7 6 5 4 3 2 1
```

```
> > > print('Starting in ', end=''); pyautogui.countdown(3)
```

```
Starting in 3 2 1
```

Estos consejos te ayudarán a hacer tus *scripts* de automatización de GUI más fáciles de usar y más capaces de recuperarse de situaciones imprevistas.

Repaso de las funciones de Pyautogui

Dado que este capítulo ha explicado muchas funciones diferentes, aquí tienes un resumen rápido a modo de referencia:

- `moveTo(x, y)`: Mueve el cursor del ratón a las coordenadas `x` e `y` dadas.

- `move(xOffset, yOffset)`: Mueve el cursor del ratón en relación a su posición actual.
- `dragTo(x, y)`: Mueve el cursor del ratón mientras el botón izquierdo está pulsado.
- `drag(xOffset, yOffset)`: Mueve el cursor del ratón en relación a su posición actual mientras el botón izquierdo está pulsado.
- `click(x, y, button)`: Simula un clic (con el botón izquierdo, por defecto).
- `rightClick()`: Simula un clic con el botón derecho.
- `middleClick()`: Simula un clic con el botón central.
- `doubleClick()`: Simula un doble clic con el botón izquierdo.
- `mouseDown(x, y, button)`: Simula la pulsación del botón especificado en la posición x, y.
- `mouseUp(x, y, button)`: Simula la liberación del botón especificado en la posición x, y.
- `scroll(units)`: Simula la rueda de desplazamiento; un argumento positivo desplaza hacia arriba y uno negativo, hacia abajo.
- `write(message)`: Escribe los caracteres pasados en la cadena `message`.
- `write([key1, key2, key3])`: Escribe las cadenas de teclas especificadas.
- `press(key)`: Pulsa la cadena de teclas especificada.
- `keyDown(key)`: Simula la pulsación de la tecla especificada.
- `keyUp(key)`: Simula la liberación de la tecla especificada.
- `hotkey(key1, key2, key3)`: Simula la pulsación de la cadena de teclas especificada y la liberación de las mismas en orden inverso.
- `screenshot()`: Devuelve una captura de pantalla como objeto `Image`

(véase el capítulo 21 para más información sobre estos objetos).

- `getActiveWindow()`, `getAllWindows()`, `getWindowsAt()` y `getWindowsWithTitle()`: Devuelve objetos `Window` que pueden redimensionar y reposicionar ventanas en el Escritorio.

- `getAllTitles()`: Devuelve una lista de cadenas del texto de la barra de título de todas las ventanas del Escritorio.

Mostrar cuadros de mensaje

Los programas que hemos escrito hasta ahora tienden a utilizar salida (con la función `print()`) y entrada (con `input()`) de texto simple. Sin embargo, los programas `PyAutoGUI` utilizarán todo tu Escritorio como su campo de acción. La ventana basada en texto en la que se ejecuta tu programa, ya sea `Mu` o una ventana del terminal, probablemente se perderá a medida que tu programa `PyAutoGUI` haga clic e interactúe con otras ventanas. Esto puede dificultar la obtención de entradas y salidas del usuario si la ventana `Mu` o la ventana del terminal quedan ocultas bajo otras ventanas.

Para resolver esto, `PyAutoGUI` incluye el módulo `PyMsgBox` para crear notificaciones emergentes para el usuario y recibir entrada. Hay cuatro funciones de cuadro de mensaje:

- `pyautogui.alert(text)`: Muestra `text` y tiene un único botón `OK`.

- `pyautogui.confirm(text)`: Muestra `text` y tiene botones `OK` y `Cancel`, devolviendo `'OK'` o `'Cancel'` dependiendo del botón pulsado.

- `pyautogui.prompt(text)`: Muestra `text` y tiene un campo de texto para que el usuario escriba, que devuelve como una cadena.

- `pyautogui.password(text)`: Es igual que `prompt()`, pero muestra asteriscos para que el usuario pueda introducir información confidencial, como una contraseña.

Estas funciones son idénticas a las cuatro descritas en “Mensajes emergentes con `PyMsgBox`” en el capítulo 12.

Resumen

La automatización de GUI con el paquete PyAutoGUI nos permite interactuar con las aplicaciones del ordenador controlando el ratón y el teclado. Aunque este enfoque es lo suficientemente flexible como para hacer todo lo que puede hacer un usuario humano, la desventaja es que estos programas son bastante ciegos a lo que están haciendo clic o escribiendo. Al escribir programas de automatización de GUI, intenta asegurarte de que fallen rápidamente si reciben instrucciones incorrectas. El fallo es molesto, pero es mucho peor que el programa continúe con el error.

Con PyAutoGUI, podemos mover el cursor del ratón por la pantalla y simular clics, pulsaciones de teclas y atajos de teclado. El paquete PyAutoGUI también puede comprobar los colores de la pantalla, lo que puede dar a nuestro programa de automatización de GUI una idea suficiente del contenido de la pantalla para saber si sigue por el buen camino. Incluso podemos dar a PyAutoGUI una captura de pantalla y dejar que calcule las coordenadas del área en la que queremos hacer clic.

Podemos combinar todas estas funciones de PyAutoGUI para automatizar cualquier tarea repetitiva y mecánica de un ordenador. De hecho, puede resultar francamente hipnótico ver cómo el cursor del ratón se mueve por sí solo y cómo aparece texto en la pantalla automáticamente. ¿Por qué no dedicar el tiempo ahorrado a sentarnos y ver cómo el programa hace todo el trabajo por nosotros? Ver cómo el ingenio nos ha librado de las tareas aburridas proporciona cierta satisfacción.

Preguntas prácticas

1. ¿Cómo podemos activar el modo seguro de PyAutoGUI para detener un programa?
2. ¿Qué función devuelve la resolución de pantalla actual?
3. ¿Qué función devuelve las coordenadas de la posición actual del cursor del ratón?
4. ¿Cuál es la diferencia entre `pyautogui.moveTo()` y

`pyautogui.move()`?

5.¿Qué funciones sirven para arrastrar el ratón?

6.¿Qué llamada a función escribe los caracteres de “Hello, world!”?

7.¿Cómo podemos pulsar teclas especiales, como la flecha izquierda del teclado?

8.¿Cómo podemos guardar el contenido actual de la pantalla en un archivo de imagen llamado `screenshot.png`?

9.¿Qué código configuraría una pausa de dos segundos después de cada llamada a una función de PyAutoGUI?

10.Si queremos automatizar clics y pulsaciones de teclado en un navegador web, ¿deberíamos usar PyAutoGUI o Selenium?

11.¿Qué hace que PyAutoGUI sea propenso a errores?

12.¿Cómo podemos averiguar el tamaño de cada ventana de la pantalla que incluya “Notepad” en el título?

13.¿Cómo podemos hacer que, por ejemplo, la ventana del navegador Firefox esté activa encima de cualquier otra ventana de la pantalla?

Programas de práctica

Para practicar, escribe programas que realicen las siguientes tareas.

Parecer ocupado

Muchos programas de mensajería instantánea determinan si estás inactivo o lejos de tu ordenador detectando la falta de movimiento del ratón durante un periodo de tiempo determinado, por ejemplo, 10 minutos. Quizás estés lejos de tu ordenador, pero no quieras que los demás vean que tu estado de mensajería instantánea pasa al modo inactivo, dando la impresión de que estás haciendo el vago. Escribe un *script* para mover el cursor del ratón un píxel hacia la izquierda cada 10 segundos y, a continuación, un píxel hacia la

derecha 10 segundos después. El movimiento debe ser pequeño y lo suficientemente infrecuente como para que no te moleste si necesitas utilizar el ordenador mientras se ejecuta el *script*.

Leer campos de texto con el portapapeles

Aunque se pueden enviar pulsaciones de teclas a los campos de texto de una aplicación con `pyautogui.write()`, no se puede utilizar `PyAutoGUI` por sí solo para leer el texto que ya se encuentra dentro de un campo de texto. Aquí es donde el módulo `pyperclip` puede ser de ayuda. Podemos utilizar `PyAutoGUI` para obtener la ventana de un editor de texto como Mu o el Bloc de notas, traerla al frente de la pantalla haciendo clic en ella, hacer clic dentro del campo de texto y, a continuación, enviar el atajo **Control-E** o **Comando-A** para seleccionar todo y **Control-C** o **Comando-C** para copiar al portapapeles. Después, el *script* de Python puede leer el texto del portapapeles ejecutando `import pyperclip` y `pyperclip.paste()`.

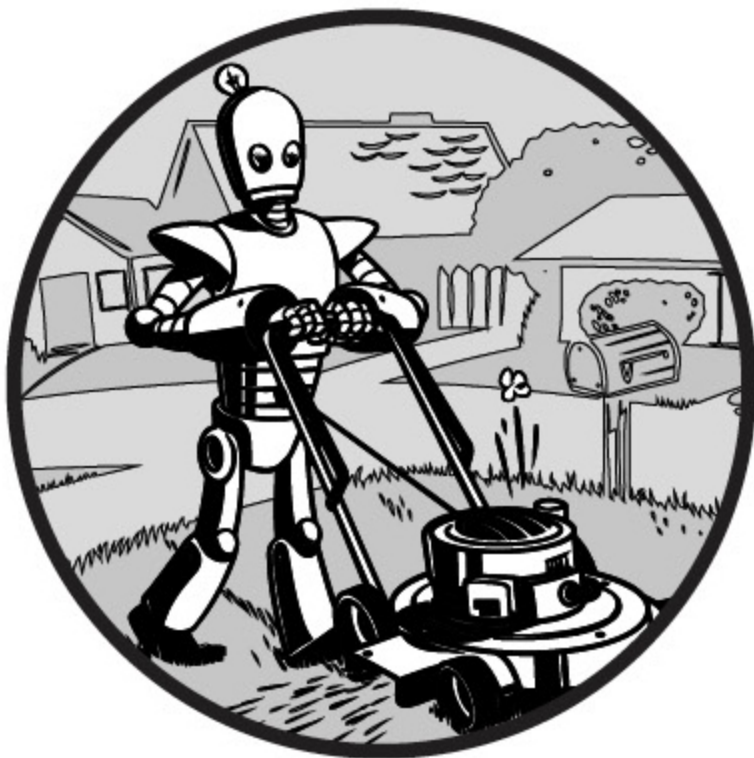
Escribe un programa que siga este procedimiento para copiar el texto de los campos de texto de una ventana. Utiliza `pyautogui.getWindowsWithTitle('Notepad')` (o el editor de texto que quieras) para obtener un objeto `Window`. Los atributos `top` y `left` de este objeto `Window` te indican dónde se encuentra la ventana, mientras que el método `activate()` se asegura de que esté en primer plano en la pantalla. A continuación, puedes hacer clic en el campo de texto principal del editor de texto añadiendo, por ejemplo, 100 o 200 píxeles a los valores de los atributos `top` y `left` con `pyautogui.click()` para colocar el foco del teclado allí. Llama a `pyautogui.hotkey('ctrl', 'a')` y `pyautogui.hotkey('ctrl', 'c')` para seleccionar todo el texto y copiarlo al portapapeles. Por último, llama a `pyperclip.paste()` para recuperar el texto del portapapeles y pegarlo en tu programa de Python. A partir de ahí, puedes utilizar esta cadena como quieras, pero por ahora simplemente pásala a `print()`.

Ten en cuenta que las funciones de ventana solo funcionan en Windows a partir de la versión 1.0.0 de `PyAutoGUI`, y no en macOS o Linux.

Escribir un bot que juega

Hay un antiguo juego Flash llamado *Sushi Go Round* que consiste en hacer clic en los botones de los ingredientes correctos para completar los pedidos de sushi de los clientes. Cuanto más rápido completes los pedidos sin cometer errores, más puntos consigues. Es una tarea perfecta para un programa de automatización de GUI, ¡y una forma de hacer trampa para conseguir una puntuación alta! Aunque Flash ya no se comercializa como producto, en <https://github.com/asweigart/sushigoroundbot> hay instrucciones para jugar sin conexión en tu ordenador y una lista de sitios web que alojan el juego *Sushi Go Round*. Ese repositorio de GitHub también tiene el código fuente de Python para un bot que juega al juego. En https://youtu.be/lfk_T6VKhTE hay un vídeo del bot jugando.

Motores de texto a voz y reconocimiento de voz



Este capítulo trata sobre un paquete de conversión de texto a voz, `pyttsx3`, y un paquete de reconocimiento de voz, `Whisper`. Los paquetes de texto a voz pueden convertir cadenas de texto en palabras habladas y, después, enviarlas a los altavoces del ordenador o guardarlas en un archivo de sonido. Al añadir esta nueva dimensión a los programas, se libera al usuario de tener que leer el texto en la pantalla. Por ejemplo, una aplicación de recetas de cocina podría leer en voz alta la lista de ingredientes mientras nos movemos por la cocina, y un *script* diario podría recopilar artículos de noticias (o correos electrónicos) y luego preparar un MP3 para reproducirlo durante el trayecto matutino al trabajo.

Por otro lado, las tecnologías de reconocimiento de voz pueden convertir archivos de audio de palabras habladas en valores de cadenas de texto. Podemos utilizar esta capacidad para añadir comandos de voz a nuestros programas o automatizar la transcripción de pódcasts. Y, a diferencia de lo que ocurre con los seres humanos, siempre se puede silenciar a un ordenador que habla demasiado.

Tanto pyttsx3 como Whisper son de uso gratuito y no requieren conexión a Internet. Los motores de texto a voz y de reconocimiento de voz que se presentan en este capítulo no se limitan al inglés, sino que funcionan con la mayoría de los idiomas humanos más hablados.

Motor de texto a voz

Para producir sonido hablado, el paquete de terceros pyttsx3 utiliza el motor de texto a voz integrado en nuestro sistema operativo: Microsoft Speech API (SAPI5) en Windows, NSSpeechSynthesizer en macOS y eSpeak en Linux. En Linux, es posible que tengas que instalar el motor ejecutando `sudo apt install espeak` desde una ventana del terminal. Puedes instalar pyttsx3 ejecutando `pip install pyttsx3` desde un terminal. El apéndice A contiene instrucciones completas para instalar paquetes de terceros.

El nombre del paquete viene de `py` para Python, `tts` para texto a voz, `x` porque es una extensión del paquete `pytts` original y `3` porque es para Python 3.

Generar voz

Producir una voz computarizada es un tema complejo en informática. Por suerte, el motor de texto a voz del sistema operativo hace el trabajo difícil por nosotros y la interacción con este motor es sencilla. Abre el editor de archivos, escribe lo siguiente y guárdalo como `hello_tts.py`:

```
import pyttsx3

engine = pyttsx3.init()
```

```
engine.say('Hello. How are you doing?')
```

```
engine.runAndWait() # El ordenador habla.
```

```
feeling = input('>')
```

```
engine.say('Yes. I am feeling ' + feeling + ' as well.')
```

```
engine.runAndWait() # El ordenador habla otra vez.
```

Después de importar el módulo `pyttsx3`, llamamos a la función `pyttsx3.init()` para inicializar el motor de voz. Esta función devuelve un objeto `Engine`. Podemos pasar una cadena de texto a su método `say()` para indicar al motor qué decir, pero la voz no saldrá hasta que llamemos al método `runAndWait()`. Este método se bloquea (es decir, no vuelve) hasta que el ordenador haya terminado de decir toda la cadena.

El programa no produce salida de texto porque nunca llama a la función `print()`. En su lugar, deberíamos oír al ordenador decir “Hello. How are you doing?”. (Asegúrate de que no está silenciado). El usuario puede escribir una respuesta con el teclado, a la que el ordenador replicará oralmente “Yes. I am feeling < tu respuesta > as well”.

El módulo `pyttsx3` permite hacer algunos cambios en la voz del ordenador. Podemos pasar las cadenas ‘rate’, ‘volume’ y ‘voices’ al método `getProperty()` del objeto `Engine` para ver su configuración actual. Introduce lo siguiente en el intérprete de comandos interactivo:

```
> > > import pyttsx3
```

```
> > > engine = pyttsx3.init()
```

```
> > > engine.getProperty('volume')
```

```
1.0
```

```
> > > engine.getProperty('rate')
```

```
200
```

```
> > > engine.getProperty('voices')
```

```
[ <pytttsx3.voice.Voice object at 0x0000029DA7FB4B10> ,  
<pytttsx3.voice.Voice object at 0x0000029DAA3DAAD0> ]
```

Ten en cuenta que la salida puede ser distinta en tu ordenador. El ajuste del volumen es un flotante, donde 1.0 indica el 100 %. La voz del ordenador habla a una velocidad de 200 palabras por minuto. Continúa el ejemplo con este código:

```
> > > for voice in engine.getProperty('voices'): # Enumera  
todas las voces disponibles.
```

```
... print(voice.name, voice.gender, voice.age, voice.languages)
```

```
.
```

```
Microsoft David Desktop - English (United States) None None []
```

```
Microsoft Zira Desktop - English (United States) None None []
```

En mi portátil Windows con idioma inglés (Estados Unidos), `getProperty('voices')` devuelve dos objetos `Voice`. (Fíjate en que la cadena está en plural, 'voices', no en singular, 'voice'). Estos objetos `Voice` tienen atributos `name`, `gender` y `age`, aunque `gender` y `age` están configurados como `None` cuando el sistema operativo no guarda esa información. El atributo `languages` es una lista de los idiomas compatibles con la voz y será una lista vacía si se desconoce esta información.

Sigamos con el ejemplo del intérprete de comandos interactivo llamando al método `setProperty()` para cambiar estos ajustes:

```
> > > engine.setProperty('rate', 300)
```

```
> > > engine.setProperty('volume', 0.5)
```

```
> > > voices = engine.getProperty('voices')
```

```
> > > engine.setProperty('voice', voices[1].id)
```

```
> > > engine.say('The quick brown fox jumps over the yellow  
lazy dog.')
```

```
> > > engine.runAndWait()
```

En este ejemplo, hemos cambiado la velocidad a 300 palabras por minuto y hemos bajado el volumen a la mitad pasando 0.5 para 'volume'. Después, hemos cambiado la voz a la femenina "Zira" que ofrece Windows pasando el atributo id del objeto Voice en el índice 1 de la lista que devuelve getProperty('voices'). Observa también que, para configurar la voz, hemos usado la cadena 'voice', en singular, no 'voices' en plural.

Guardar sonido en archivos WAV

El método save_to_file() del módulo pytsx3 puede guardar el sonido generado en un archivo WAV (con extensión .wav). Escribe lo siguiente en el intérprete de comandos interactivo:

```
> > > import pytsx3
```

```
> > > engine = pytsx3.init()
```

```
> > > engine.save_to_file('Hello. How are you doing?',  
'hello.wav')
```

```
> > > engine.runAndWait() # El ordenador crea hello.wav.
```

El primer argumento para save_to_file() es una cadena con el discurso que se genera, mientras que el segundo es el nombre del archivo .wav. La cadena de texto puede ser una oración corta, como en el ejemplo, o páginas enteras de texto. En mi ordenador, pytsx3 fue capaz de convertir una cadena de 1.800 palabras en un archivo de audio de 10 minutos en unos dos segundos. Es importante destacar que solo llamar a save_to_file() no es suficiente, también hay que llamar al método runAndWait() para que Python cree el archivo .wav. El módulo pytsx3 puede guardar solo archivos .wav, no .mp3 ni otros formatos de audio.

Reconocimiento de voz

Whisper es un sistema de reconocimiento de voz que puede reconocer varios idiomas. A partir de un archivo de audio o vídeo, Whisper puede devolver el discurso como texto en una cadena de Python. También devuelve los tiempos de inicio y fin de grupos de palabras, que se pueden utilizar para generar archivos de subtítulos.

Instala Whisper ejecutando `pip install openai-whisper` desde el terminal. (Observa que el nombre del paquete de reconocimiento de voz es `openai-whisper`; el paquete `whisper` del sitio web PyPI es otra cosa). Es una descarga de gran tamaño y la instalación puede tardar varios minutos. Además, la primera vez que llamemos a la función `load_model()`, el ordenador descargará el modelo de reconocimiento de voz, que puede tener un tamaño de cientos de megabytes o más.

Supongamos que tenemos un archivo de audio llamado `hello.wav` en el directorio de trabajo actual. (Whisper también puede manejar `.mp3` y varios otros formatos de audio). Podríamos introducir lo siguiente en el intérprete de comandos interactivo:

```
> > > import whisper  
  
> > > model = whisper.load_model('base')  
  
> > > result = model.transcribe('hello.wav')  
  
> > > print(result['text'])
```

Hello. How are you doing?

Después de importar el módulo `whisper`, hay que cargar el modelo de reconocimiento de voz llamando a la función `whisper.load_model()` y pasándole la cadena del modelo de aprendizaje automático entrenado que queremos usar: `'tiny'`, `'base'`, `'small'`, `'medium'` o `'large-v3'`. (Seguirán saliendo nuevos modelos). Los modelos más pequeños pueden transcribir más deprisa, pero los más grandes transcriben con mayor precisión, incluso cuando el audio tiene sonido ambiente de fondo. La primera vez que cargues un modelo, tu ordenador debe estar conectado a Internet para que el módulo `whisper` pueda descargarlo de los servidores de OpenAI. La tabla 24.1 recoge los nombres de los modelos como cadenas que puedes pasar a la función `whisper.load_model()`, junto con su

tamaño de archivo, uso de memoria y los resultados de algunas pruebas de tiempo de ejecución en mi ordenador portátil.

Tabla 24.1. Propiedades de los modelos de reconocimiento de voz de Whisper.

Modelo **Tamaño de archivo** **Uso de memoria** **Resultados** **Pruebas de tiempo de ejecución**

base 15MB 15MB 15MB 15MB

medium 35MB 35MB 35MB 35MB

large-v3 35MB 35MB 35MB 35MB

large-v3-turbo 15MB 15MB 15MB 15MB

large-v3-turbo 15MB 15MB 15MB 15MB

Mi opinión personal es que, para el 99 % de los casos, el modelo ‘base’ debería ser adecuado y el modelo ‘medium’ es suficiente cuando se necesita mayor precisión. Todos los modelos producen errores, por lo que el resultado siempre debe ser revisado por un humano. Puedes probar con modelos más grandes si encuentras muchos errores de transcripción en el texto, o con modelos más pequeños si Whisper tarda demasiado en transcribir el audio. Sin embargo, como se aprecia en la tabla 24.1, hay una diferencia sustancial entre los dos minutos y 34 segundos que tarda el modelo ‘base’ en transcribir 15 minutos de audio y los casi 33 que tarda el modelo ‘large-v3’.

Con el objeto `model.Whisper` que devuelve `whisper.load_model()`, podemos llamar al método `transcribe()` para realizar la transcripción real. Pasa al método una cadena con el nombre del archivo de audio. Este método tardará entre unos segundos y unas horas en ejecutarse, dependiendo del modelo y la duración del archivo de audio. Whisper puede aceptar cualquier archivo de audio o vídeo y lo convierte automáticamente al formato que requiere.

Whisper puede detectar automáticamente el idioma del audio, pero podemos especificar una lengua pasando un argumento de idioma para transcribir, como `model.transcribe('hello.wav', language='English')`. Para encontrar los idiomas que admite Whisper, podemos ejecutar `whisper --help` desde el terminal. Whisper es bastante bueno (aunque no perfecto) a la hora de adivinar dónde debe insertar la puntuación y poner en mayúscula

los nombres propios. Sin embargo, siempre debes revisar el resultado para corregir cualquier error.

El diccionario que devuelve `model.transcribe()` tiene varios pares clave-valor, pero la clave 'text' contiene la cadena de la transcripción.

Por defecto, Whisper utiliza la CPU para transcribir texto, pero, si el ordenador tiene una tarjeta gráfica 3D, se puede acelerar considerablemente las transcripciones configurándolo para utilizar la unidad de procesamiento gráfico (GPU). Encontrarás estas instrucciones de configuración en la documentación en línea de <https://github.com/openai/whisper>. Si tu ordenador tiene una tarjeta gráfica NVIDIA, puedes seguir las instrucciones del apéndice A para instalar paquetes para reconocimiento de voz más rápido. Para utilizar la GPU, sustituye el código `whisper.load_model('base')` de este capítulo por `whisper.load_model('base', device='cuda')`.

Encontrarás varias opciones adicionales para Whisper en su documentación en línea.

Crear archivos de subtítulo

Además del audio transcrito, el diccionario de resultados de Whisper contiene información temporal que identifica la ubicación del texto en el archivo de sonido. Podemos utilizar este texto y estos datos temporales para generar archivos de subtítulos que otros programas puedan importar. Los dos formatos de archivo de subtítulos más comunes son SRT SubRip Subtitle (con la extensión .srt) y VTT Web Video Text Tracks (con la extensión .vtt). SRT es un estándar más antiguo y extendido, mientras que los sitios web de vídeo modernos suelen utilizar VTT. Los formatos son similares. Por ejemplo, aquí está la primera parte de un archivo SRT:

1

00:00:00,000 --> 00:00:05,640

Dinosaurs are a diverse group of reptiles of the clade dinosauria.
They first

00:00:05,640 --> 00:00:14,960

appeared during the triassic period. Between 245 and 233.23 million years ago.

--*fragmento omitido*--

Compárala con la primera parte de un archivo VTT para los mismos subtítulos:

WEBVTT

00:00.000 --> 00:05.640

Dinosaurs are a diverse group of reptiles of the clade dinosauria. They first

00:05.640 --> 00:14.960

appeared during the triassic period. Between 245 and 233.23 million years ago.

--*fragmento omitido*--

Ambos archivos indican que las palabras “Dinosaurs are a diverse group...” aparecen entre el inicio del archivo de audio transcrito (en el segundo 0) y el segundo 5,640.

Whisper también puede generar sus resultados como datos TSV (con la extensión .tsv) o datos JSON (con la extensión .json). TSV no es un formato oficial de subtítulos, pero puede resultar útil si necesitas exportar el texto y los datos de sincronización a, por ejemplo, otro programa de Python que pueda leerlos utilizando el módulo csv que vimos en el capítulo 18. Los subtítulos con formato TSV tienen el siguiente aspecto:

~~start~~

00:00:05,640 Dinosaurs are a diverse group of reptiles of the clade dinosauria. They

00:00:14,960 appeared during the triassic period. Between 245 and 233.23

million years ago.

--fragmento omitido --

Para crear estos archivos de subtítulos, añade dos líneas adicionales de código después de llamar a `model.transcribe()`:

```
> > > import whisper

> > > model = whisper.load_model('base')

> > > result = model.transcribe('hello.wav')

1 > > > write_function = whisper.utils.get_writer('srt', )

2 > > > write_function(result, 'audio')
```

La función `whisper.utils.get_writer()` 1 acepta el formato de archivo de subtítulos como una cadena ('srt', 'vtt', 'txt', 'tsv' o 'json') y la carpeta en la que guardar el archivo (con la cadena '.' indicando el directorio de trabajo actual). La función `get_writer()` devuelve una función para pasar los resultados de la transcripción. (Es una forma bastante extraña de crear los archivos de transcripción, pero es así como está diseñado el módulo `whisper`). La almacenamos en una variable llamada `write_function`, que después podemos tratar como una función y llamada, pasando el diccionario `result` y el nombre para el archivo de subtítulos 2. Estas dos líneas de código producen un archivo con formato SRT llamado `audio.srt` en el directorio de trabajo actual, utilizando la información de texto y tiempo del diccionario `result`.

Descargar vídeos de sitios web

Aunque descargar archivos de audio para transcribir con el reconocimiento de voz de `Whisper` suele ser fácil, hay sitios web de vídeo, como YouTube, que a menudo dificultan la descarga de contenido. El módulo `yt-dlp` permite a los *scripts* de Python descargar vídeos de YouTube y centenares de otros sitios web de vídeo para poder verlos sin conexión. El apéndice A contiene instrucciones para instalar `yt-dlp`. Una vez instalado, el siguiente código descargará el vídeo de la URL especificada:

```
> > > import yt_dlp
```

```
> > > video_url = 'https://www.youtube.com/watch?v=kSrnlbioN6w'
```

```
> > > with yt_dlp.YoutubeDL() as ydl:
```

```
... ydl.download([video_url])
```

```
...
```

Fíjate en que la función `ydl.download()` espera una lista de URL de vídeos, por lo que colocamos la cadena `video_url` dentro de una lista antes de pasarla a la llamada a la función. El nombre del archivo del vídeo se basa en el título del sitio web del vídeo y puede tener la extensión `.mp4`, `.mkv` u otra extensión de formato de vídeo. Verás muchos mensajes de depuración mientras se descarga el vídeo.

El sitio web de vídeo podría rechazar la descarga debido a requisitos de edad o de inicio de sesión, restricciones geográficas o medidas contra el raspado web. Si encuentras errores, lo primero que debes hacer es instalar la última versión de `yt-dlp`, que se actualiza para mantener la compatibilidad a medida que los sitios web de vídeo cambian su diseño.

Puedes leer sobre las numerosas opciones de configuración de `yt-dlp` en la documentación en línea en <https://pypi.org/project/yt-dlp/>. Por ejemplo, se puede extraer el sonido de un vídeo de YouTube pasando un diccionario de ajustes de configuración a la función `yt_dlp.YoutubeDL()`:

```
> > > import yt_dlp
```

```
> > > video_url = 'https://www.youtube.com/watch?v=kSrnlbioN6w'
```

```
> > > options = {
```

```
. 1 'quiet': True, # Suprime la salida.
```

```
... 'no_warnings': True, # Suprime las advertencias.
```

```
. 2 'outtmpl': 'downloaded_content.%(ext)s',
... 'format': 'm4a/bestaudio/best',
... 'postprocessors': { # Extrae el audio con ffmpeg.
. 'key': 'FFmpegExtractAudio',
. 'preferredcodec': 'm4a',
... ]
. }
.
> > > with yt_dlp.YoutubeDL(options) as ydl:
... ydl.download( [video_url] )
...
```

Los pares clave-valor 'quiet': True y 'no_warnings': True 1 evitan la salida verbosa de la depuración. El diccionario options pasado a yt_dlp.Youtube() le dice que descargue el vídeo y extraiga el audio a un archivo llamado downloaded_content.m4a 2. (La extensión de archivo puede variar si el vídeo tiene un formato de audio diferente, pero .m4a es el formato más popular). Si no hubiéramos configurado el par clave-valor 'outtmpl': 'downloaded_content.%(ext)s' en el diccionario options, el nombre del archivo descargado se basaría en el título del vídeo (excluyendo los caracteres no permitidos en nombres de archivo, como interrogantes y dos puntos).

Para obtener el nombre exacto del archivo, podemos usar los patrones globales explicados en el capítulo 10. Sabemos que la parte principal del archivo es 'downloaded_content', pero la extensión podría ser cualquier formato de audio. El siguiente código utiliza objetos Path para averiguar el nombre exacto del archivo descargado:

```
> > > from pathlib import Path
```

```
> > > matching_filenames =
list(Path().glob('downloaded_content.*'))

> > > downloaded_filename = str(matching_filenames[0])

> > > downloaded_filename

'downloaded_content.m4a'
```

Establecer el nombre de archivo hace que al código le resulte más fácil usar el archivo para, por ejemplo, ejecutarlo con el reconocimiento de voz de Whisper. Los modelos 'base' y 'medium' crean subtítulos de mucha más calidad que los subtítulos automáticos que genera YouTube actualmente.

Si solo quieres descargar la información sobre un vídeo determinado, puedes decir a yt-dlp que omita el archivo y descargue solo sus metadatos con el siguiente código:

```
> > > import yt_dlp, json

> > > video_url = 'https://www.youtube.com/watch?
v=kSrnlbioN6w'

> > > options = {

... 'quiet': True, # Suprime la salida.

... 'no_warnings': True, # Suprime las advertencias.

. 1 'skip_download': True, # No descarga el vídeo.

. }

.

> > > with yt_dlp.YoutubeDL(options) as ydl:

. 2 info = ydl.extract_info(video_url)

. 3 json_info = ydl.sanitize_info(info)

... print('TITLE:', json_info['title']) # Imprime el título del vídeo.
```

```
... print('KEYS:', json_info.keys())

... with open('metadata.json', 'w', encoding='utf-8') as
json_file:
```

```
. 4 json_file.write(json.dumps(json_info))
```

TITLE: Beyond the Basic Stuff with Python - Al Sweigart - Part 1

KEYS: dict_keys(['id', 'title', 'formats', 'thumbnails', 'thumbnail',
'description', 'channel_id', 'channel_url', 'duration', 'view_count',
'average_rating', 'age_limit', 'webpage_url',

--fragmento omitido--

Si solo queremos los metadatos del vídeo y no el vídeo en sí, podemos incluir el par clave-valor 'skip_download': True 1 en el diccionario options pasado a yt_dlp.YoutubeDL(). La llamada al método ydl.extract_info() devuelve un diccionario de información sobre el vídeo 2. Algunos de estos datos podrían no tener el formato correcto como JSON (explicado en el capítulo 18), pero podemos ponerlos en una forma compatible con JSON llamando a ydl.sanitize() 3. El diccionario que devuelve el método sanitize() tiene varias claves, incluidas 'title' para el nombre del vídeo, 'duration' para su duración en segundos, etc. Para rematar, nuestro código escribe estos datos JSON en un archivo llamado metadata.json 4.

Resumen

Una de las mayores fortalezas de Python es su amplio ecosistema de paquetes de terceros para tareas como la conversión de texto a voz y el reconocimiento de voz. Estos paquetes cogen algunos de los problemas más difíciles de la informática y los ponen a disposición de nuestros programas con solo unas pocas líneas de código.

El paquete pyttsx3 convierte texto en voz utilizando el motor de voz

del ordenador para crear audio que se puede reproducir por los altavoces o guardar en un archivo .wav. El sistema de reconocimiento de voz Whisper utiliza varios modelos subyacentes para transcribir las palabras de un archivo de audio. Estos modelos tienen diferentes tamaños; los modelos más pequeños transcriben más rápido con menos precisión, mientras que los modelos más grandes son más lentos, pero más exactos. Funcionan para muchas lenguas naturales, no solo para el inglés. Whisper se ejecuta en el ordenador y no se conecta a servidores en línea, excepto para descargar el modelo la primera vez que se utiliza.

Los motores de voz que utilizan estos paquetes de Python han experimentado un gran salto en calidad que no estaba disponible antes de la década de 2020. Python es un excelente “lenguaje pegamento” que permite que nuestros *scripts* se conecten con este software para que podamos añadir funciones de voz a nuestros propios programas con solo unas pocas líneas de código. Si quieres aprender más sobre el texto a voz y el reconocimiento de voz, encontrarás muchos proyectos de ejemplo divertidos en *Make Python Talk*, de Mark Liu (No Starch Press, 2021).

Preguntas prácticas

1. ¿Cómo podemos hacer que la voz de pytsx3 hable más rápido?
2. ¿En qué formato guarda el audio pytsx3?
3. ¿Dependen pytsx3 y Whisper de servicios en línea?
4. ¿Son compatibles pytsx3 y Whisper con otras lenguas, además del inglés?
5. ¿Cómo se llama el modelo de aprendizaje automático predeterminado de Whisper para el reconocimiento de voz?
6. Nombra dos formatos de archivos de subtítulos habituales.
7. ¿Puede yt-dlp descargar vídeos de sitios distintos de YouTube?

Programas de práctica

Para practicar, escribe programas que hagan las siguientes tareas.

Añadir voz a Adivina el número

Vuelve al juego de adivinación del capítulo 3 y añádele una función de voz. Sustituye todas las llamadas a `print()` por llamadas a una función llamada `speak()`. A continuación, define la función `speak()` para que acepte un argumento de cadena (como hacía `print()`), pero haz que imprima la cadena en pantalla y además la diga en voz alta. Por ejemplo, puedes sustituir esta línea de código:

```
print('I am thinking of a number between 1 and 20.')
```

por esta otra:

```
speak('I am thinking of a number between 1 and 20.')
```

Para aprovechar al máximo la función de generación de voz, vamos a cambiar el texto 'Your guess is too low.' y 'Your guess is too high.' para que diga el intento del jugador. Por ejemplo, el ordenador debería decir: "Your guess, 42, is too low". También puedes añadir voz a otros proyectos de este libro, como el juego de piedra, papel o tijera.

Cantar 99 Bottles of Beer

Las canciones acumulativas son aquellas cuyos versos se repiten con adiciones o pequeños cambios. *99 Bottles of Beer*, *The 12 Days of Christmas* o *Cu-cú cantaba la rana* son ejemplos de canciones acumulativas. Escribe un programa que cante (o al menos diga) la letra de *99 Bottles of Beer*:

99 bottles of beer on the wall,

99 bottles of beer,

Take one down, pass it around,

98 bottles of beer on the wall.

Esta letra se repite con una botella menos cada vez y sigue hasta

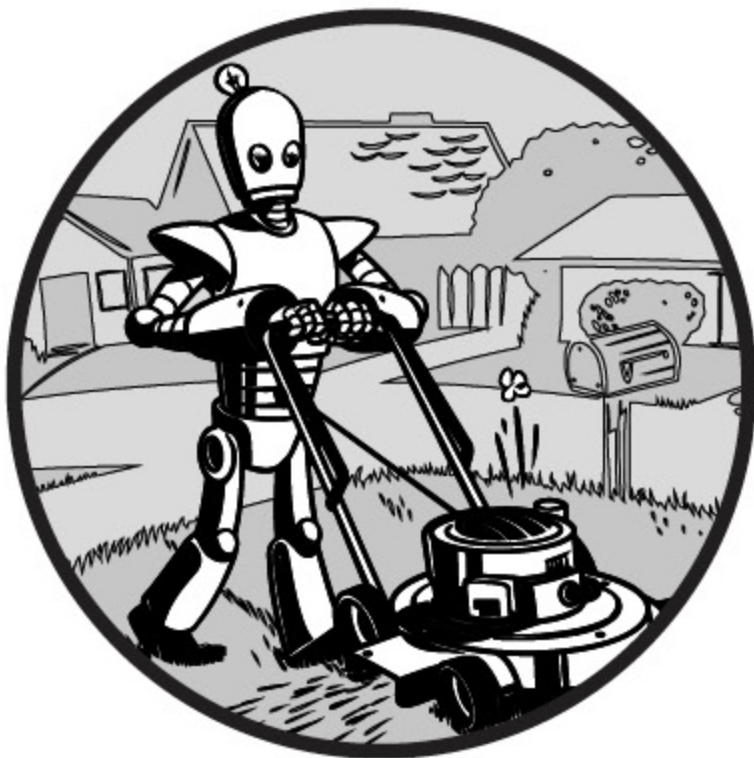
llegar a cero botellas, cuando dice “No more bottles of beer on the wall”. (Puede que te convenga que el programa empiece con 2 o 3 botellas en vez de 99 para que sea más fácil hacer las pruebas).

Transcriptor de YouTube

Escribe un programa que aglutine las funciones de yt-dlp y Whisper para descargar automáticamente vídeos de YouTube y producir archivos de subtítulos en formato .srt. La entrada puede ser una lista de URL para descargar y transcribir. También puedes añadir opciones para producir distintos formatos de subtítulos. Python es un excelente “lenguaje pegamento” para combinar las capacidades de distintos módulos.

A

Instalar paquetes de terceros



Muchos desarrolladores han escrito sus propios módulos, aumentando las capacidades de Python más allá de las que ofrecen las librerías estándar que vienen con él. La principal manera de acceder a paquetes de terceros es usar la herramienta `pip` de Python, que descarga con seguridad e instala módulos en el ordenador desde el sitio web Python Package Index (PyPI), una especie de repositorio de módulos para Python gratuitos. Este apéndice incluye instrucciones para instalar los paquetes que se usan a lo largo del libro.

Para obtener información general sobre el trabajo en la línea de

comandos, los entornos virtuales, los paquetes y los módulos, consulta el capítulo 12.

Instalar pip

El archivo ejecutable para la herramienta pip se llama pip en Windows y pip3 en macOS y Linux. Esta herramienta se incluye en Python desde la versión 3.4. Sin embargo, algunas distribuciones de Linux pueden no tenerla preinstalada.

Para instalar pip3 en Ubuntu o Debian Linux, abre una ventana del terminal y escribe `sudo apt install python3-pip`. Para instalarlo en Fedora Linux, escribe `sudo yum install python3-pip`. Estos comandos te piden introducir la contraseña de administrador de tu ordenador.

Encontrar pip

Si la carpeta de pip no está incluida en la variable de entorno PATH, es posible que tengas que cambiar de directorio en el terminal con el comando `cd` antes de ejecutar pip. Primero, busca tu nombre de usuario ejecutando `echo %USERNAME%` en Windows o `whoami` en macOS y Linux.

A continuación, en Windows, ejecuta el siguiente comando, especificando tu nombre de usuario y ajustando el nombre de la carpeta para la versión de Python que tienes instalada:

```
C:\Users\al> cd C:\Users\tu_usuario\AppData\Local\Programs\Python\Python313\Scripts
```

En macOS, ejecuta este otro comando (asegurándote de especificar el nombre correcto para la carpeta):

```
al@Als-MacBook-Pro ~ % cd /Library/Frameworks/Python.framework/Versions/3.13/bin/
```

En Linux, ejecuta este comando, especificando tu nombre de usuario:

```
al@al-VirtualBox:~$ cd /home/tu_usuario/.local/bin/
```

Ya deberías estar en la carpeta correcta para ejecutar la herramienta pip. Como alternativa, puedes añadir todas estas carpetas a la variable de entorno PATH siguiendo las instrucciones del apartado “La variable de entorno PATH” del capítulo 12.

Ejecutar pip desde entornos virtuales

Algunos sistemas operativos no permiten ejecutar pip sin crear y activar antes un entorno virtual con el módulo integrado de Python venv. Para aprender Python y experimentar con código, está bien crear un entorno virtual para todos tus programas. Consulta “Entornos virtuales”, en el capítulo 12, para más detalles.

Instalar los paquetes utilizados en este libro

Dado que futuros cambios en los paquetes de terceros pueden ser incompatibles con los ejemplos de código del libro, te recomiendo instalar exactamente las versiones que he usado yo añadiendo `=` versión al final del nombre módulo. (Fíjate en los dos signos de igualdad de esta opción de línea de comandos). Por ejemplo, `pip install send2trash = 1.8.3` instala la versión 1.8.3 del paquete `send2trash`.

La manera más fácil de instalar todos los paquetes compatibles de una sola vez es instalar el paquete `automateboringstuff3`, que actualizo a medida que van saliendo nuevas versiones de paquetes de terceros, siempre que sean compatibles con los ejemplos de código del libro. Si quieres instalar las versiones más recientes, consulta su documentación en línea para información de uso actualizada.

En Windows, ejecuta el siguiente comando:

```
C:\Users\al> python -m pip install automateboringstuff3
```

En macOS y Linux, ejecuta este otro:

```
al@Als-MacBook-Pro ~ % python3 -m pip install  
automateboringstuff3
```

Para instalar el paquete PyAutoGUI del capítulo 23 en Linux, debes

dar algunos pasos adicionales. Abre una ventana del terminal y ejecuta `sudo apt install python3-tk` y `sudo apt install python3-dev`. Para que el módulo `pyperclip` del capítulo 8 funcione en Linux, debes ejecutar `sudo apt install xclip`. Necesitarás la contraseña de administrador del equipo.

En el momento de la publicación de este libro, estos comandos instalan las siguientes versiones de los paquetes:

`beautifulsoup4` == 4.12.3

`matplotlib` == 3.9.2

`openpyxl` == 3.1.5

`pdfminer.six` == 20240706

`pillow` == 10.4.0

`playsound3` == 2.4.0

`playwright` == 1.47.0

`PyPDF` == 5.0.1

`python-docx` == 1.1.2

`pyttsx3` == 2.98

`requests` == 2.32.3

`selenium` == 4.25.0

`send2trash` == 1.8.3

`xmldict` == 0.13.0

Además, el paquete `automateboringstuff3` siempre instala la última versión de estos paquetes:

`bext`

`ezgmail`

ezsheets

humre

pyautogui

pymsgbox

pyperclip

pyperclipimg

yt-dlp

El paquete Playwright requiere un paso más para instalar los navegadores que utiliza el módulo: tras instalar el paquete `automateboringstuff3`, ejecuta `playwright install`.

El paquete `pytesseract` del capítulo 22 y el paquete `openai-whisper` del capítulo 24 son bastante grandes y es posible que algunos lectores no deseen instalarlos, así que no los he incluido en el paquete `automateboringstuff3`. Puedes instalarlos ejecutando lo siguiente en Windows:

```
C:\Users\al> pip install pytesseract == 0.3.10
```

```
C:\Users\al> pip install openai-whisper == 20231117
```

En macOS y Linux, ejecuta lo siguiente:

```
al@Als-MacBook-Pro ~ % pip3 install pytesseract == 0.3.10
```

```
al@Als-MacBook-Pro ~ % pip3 install openai-whisper == 20231117
```

Si estás en Windows, podrías ver un error como este al instalar Whisper:

A module that was compiled using NumPy 1.x cannot be run in NumPy 2.0.1 as it may crash

Puedes resolver este problema ejecutando `pip install "numpy < 2"`

para devolver NumPy a una versión compatible. Asegúrate de incluir las comillas dobles.

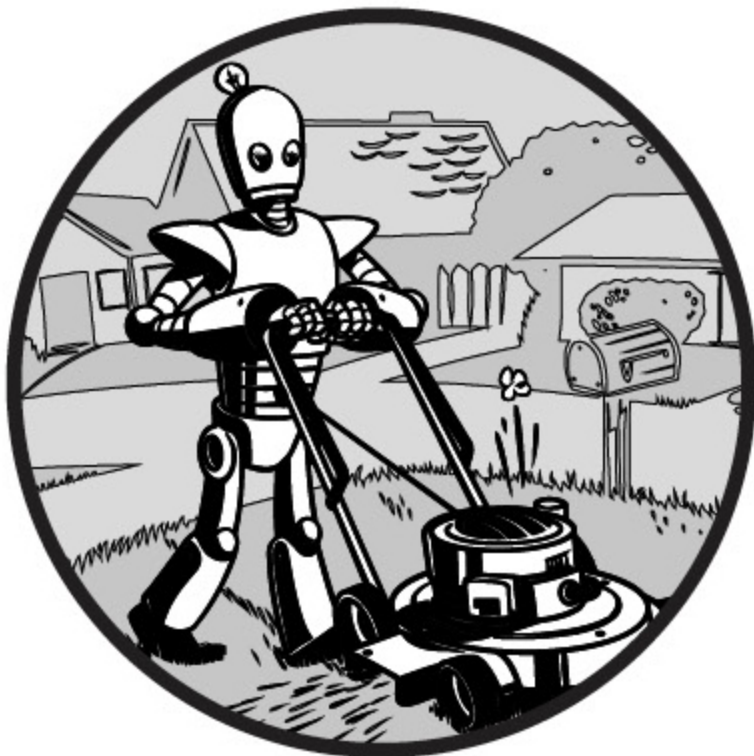
Asimismo, el paquete Whisper puede usar la tarjeta gráfica (GPU) NVIDIA de tu ordenador para realizar el reconocimiento del discurso más deprisa si ejecutas este comando:

```
C:\Users\al> pip install torch torchvision torchaudio --index-url  
https://download.pytorch.org/whl/cu118
```

Los ordenadores con GPU de marcas distintas de NVIDIA, incluidos todos los MacBooks, no son compatibles con esta función.

B

Respuestas a las preguntas prácticas



Este apéndice contiene las respuestas a las preguntas prácticas que hay al final de cada capítulo. Te recomiendo encarecidamente que te tomes tu tiempo para trabajar en estas preguntas. Programar es más que memorizar sintaxis y una lista de nombres de funciones. Igual que cuando se aprende un idioma extranjero, cuanto más se practique más rendimiento se saca. También hay muchos sitios web con ejercicios de programación.

En lo que a los programas de práctica se refiere, no hay una única solución correcta. Mientras tu programa haga lo que pedía el proyecto, puedes considerarlo correcto. Sin embargo, si quieres ver

ejemplos de proyectos completos, están disponibles en la página web del libro.

Capítulo 1

1. Los operadores son `+`, `-`, `*` y `/`. Los valores son `'hello'`, `-88.8` y `5`.
2. La cadena es `'spam'`; la variable es `spam`. Las cadenas siempre empiezan y terminan con comillas.
3. Los tres tipos de datos presentados en este capítulo son números enteros, flotantes y cadenas.
4. Una expresión es una combinación de valores y operadores. Todas las expresiones se evalúan (es decir, se reducen) a un único valor.
5. Una evaluación se evalúa como un único valor. Una declaración, no.
6. La variable `bacon` se establece en `20`. La expresión `bacon + 1` no reasigna el valor de `bacon` (eso requeriría una declaración de asignación: `bacon = bacon + 1`).
7. Ambas expresiones se evalúan como la cadena `'spamspamspam'`.
8. Los nombres de variable no pueden empezar con un número.
9. Las funciones `int()`, `float()` y `str()` se evaluarán como las versiones entera, flotante y de cadena de los valores que les pasemos.
10. La expresión da error porque `99` es un entero y solo las cadenas se pueden concatenar con otras cadenas con el operador `+`. La forma correcta es `I have eaten ' + str(99) + ' burritos.'`.

Capítulo 2

1. `True` y `False`, con `T` y `F` mayúscula y el resto de la palabra en minúsculas.
2. `and`, `or` y `not`.

3. True and True es True.

True and False es False.

False and True es False.

False and False es False.

True or True es True.

True or False es True.

False or True es True.

False or False es False.

not True es False.

not False es True.

4. False.

False.

True.

False.

False.

True.

5. ==, !=, <, >, <= y >=.

6. == es el operador de igualdad que compara dos valores y se reduce a un valor booleano, mientras que = es el valor de asignación que almacena un valor en una variable.

7. Una condición es una expresión que se usa en una declaración de control de flujo y se evalúa a un valor booleano.

8. Las líneas print('bacon') y print('ham') son bloques en sí mismas y un tercer bloque es todo lo que va entre if spam == 10: y el

```
print('Done'): final.
```

```
print('eggs')
```

```
if spam > 5:
```

```
print('bacon')
```

```
else:
```

```
print('ham')
```

```
print('spam')
```

9.El código:

```
if spam == 1:
```

```
print('Hello')
```

```
elif spam == 2:
```

```
print('Howdy')
```

```
else:
```

```
print('Greetings!')
```

Capítulo 3

1.Pulsa **Control-C** para detener un programa atrapado en un bucle infinito.

2.La declaración **break** moverá la ejecución del programa fuera del bucle, justo después. La declaración **continue** moverá la ejecución al principio del bucle.

3.Todas hacen lo mismo. La llamada a `range(10)` crea un rango de 0 hasta (pero sin incluir) 10, `range(0, 10)` dice explícitamente al bucle que empiece en 0 y `range(0, 10, 1)` dice explícitamente al bucle que aumente la variable en 1 en cada iteración.

4.El código:

```
for i in range(1, 11):
```

```
    print(i)
```

```
    y:
```

```
    i = 1
```

```
    while i <= 10:
```

```
        print(i)
```

```
        i = i + 1
```

5.Se puede llamar a esta función con `spam.bacon()`.

Capítulo 4

1.Las funciones reducen la necesidad de duplicar código. Esto hace los programas más cortos, más legibles y más fáciles de actualizar.

2.El código de una función se ejecuta cuando se llama a la función, no cuando se define.

3.La declaración `def` define (esto es, crea) una función.

4.Una función consta de la declaración `def` y el código de su proposición `def`. Una llamada a una función es lo que mueve la ejecución del programa a la función en cuestión y evalúa el valor de retorno de la función.

5.Hay un ámbito global y se crea un ámbito local cada vez que se llama a una función.

6.Cuando vuelve una función, el ámbito local se destruye y todas sus variables se olvidan.

7.Un valor de retorno es el valor al que se evalúa una llamada a una función. Como cualquier valor, un valor de retorno puede formar parte de una expresión.

- 8.Si no hay declaración return para una función, su valor de retorno es None.
- 9.Una declaración global obliga a una variable en una función a hacer referencia a la variable global.
- 10.El tipo de datos de None es NoneType.
- 11.Esa declaración import importa un módulo llamado areallyourpetsnamederic. (Por cierto, no es un módulo real de Python).
- 12.Esta función puede llamarse con spam.bacon().
- 13.Poniendo la línea de código que podría provocar el error en una proposición try. El código que se ejecuta si se produce un error va en la proposición except.
- 15.La variable global random_number se establece una vez como un número aleatorio y la variable random_number de la función get_random_dice_roll() utiliza la variable global. Esto significa que se devuelve el mismo número para cada llamada a la función random_dice_roll().

Capítulo 5

- 1.assert spam >= 10, 'The spam variable is less than 10.'
- 2.O assert eggs.lower() != bacon.lower() 'The eggs and bacon variables are the same!' o assert eggs.upper() != bacon.upper(), 'The eggs and bacon variables are the same!'
- 3.assert False, 'This assertion always triggers.'
- 4.Para poder llamar a logging.debug(), hay que tener estas dos líneas al principio del programa:
- ```
import logging

logging.basicConfig(level=logging.DEBUG, format='%(asctime)s -
%(levelname)s - %(message)s')
```

5. Para poder enviar mensajes de registro a un archivo llamado `programLog.txt` con `logging.debug()`, hay que tener estas dos líneas al principio del programa:

```
import logging
```

```
logging.basicConfig(filename = 'programLog.txt',
level = logging.DEBUG,
```

```
format = ' %(asctime)s - %(levelname)s - %(message)s')
```

6. `DEBUG`, `INFO`, `WARNING`, `ERROR`, and `CRITICAL`.

7. `logging.disable(logging.CRITICAL)`.

8. Podemos deshabilitar los mensajes de registro sin eliminar las llamadas a la función de registro. Podemos deshabilitar selectivamente los mensajes del nivel más bajo. Podemos crear mensajes de registro. Los mensajes de registro proporcionan un sello de tiempo.

9. El botón Paso dentro moverá el depurador a una llamada a función. El botón Paso por encima ejecutará rápidamente la función sin meterse en ella. El botón Paso fuera ejecutará rápidamente el resto del código hasta salir de la función en la que se encuentra en ese momento.

10. Después de hacer clic en Continuar, el depurador se detendrá cuando haya llegado al final del programa o a una línea con un punto de interrupción.

11. Un punto de interrupción es una configuración en una línea de código que hace que el depurador se detenga cuando el programa llegue ahí.

12. Para establecer un punto de interrupción en Mu, haz clic en el número de línea para que aparezca un punto rojo al lado.

## Capítulo 6

1. El valor de lista vacía, que es una lista que no contiene elementos.

Es como el valor de cadena vacía "".

2.spam[2] = 'hello' (observa que el tercer valor de la lista es un índice 2 porque el primer índice es 0).

3.'d' (observa que '3' \* 2 es la cadena '33', que pasa a int() antes de dividirse entre 11. Al final esto se evalúa a 3. Podemos usar expresiones en cualquier parte donde se use un valor).

4.'d' (los índices negativos cuentan desde el final).

5.['a', 'b']

6.1.

7.[3.14, 'cat', 11, 'cat', True, 99].

8.[3.14, 11, 'cat', True].

9.El operador para concatenar listas es +, mientras que el operador de replicación es \*. (Es igual que con las cadenas).

10.append() añade valores al final de una lista e insert() puede añadirlos en cualquier parte de la lista.

11.La declaración del y el método de lista remove() son dos formas de eliminar valores de una lista.

12.Tanto las listas como las cadenas pueden pasarse a len(), tener índices y rebanadas, usarse en bucles for, concatenarse o replicarse y utilizarse con los operadores in y not in.

13.Las listas son mutables; podemos añadir, eliminar o cambiar sus valores. Las tuplas son inmutables; no se puede cambiar nada. Además, las tuplas se escriben entre paréntesis, ( y ), mientras que las listas usan corchetes, [ y ].

14.(42,) (la coma final es obligatoria).

15.Las funciones tuple() y list(), respectivamente.

16.Contienen referencias a valores de lista.

17. La función `copy.copy()` hará una copia superficial de la lista, mientras que `copy.deepcopy()` hará una copia profunda. Es decir, solo `copy.deepcopy()` duplicará las listas dentro de una lista.

## Capítulo 7

1. Dos llaves: `{}`.

2. `{'foo': 42}`.

3. Los elementos almacenados en un diccionario están desordenados, mientras que los de una lista están ordenados.

4. Obtienes un error `KeyError`.

5. No hay diferencia. El operador `in` comprueba si existe un valor como clave en el diccionario.

6. La expresión `'cat' in spam` comprueba si hay una clave `'cat'` en el diccionario, mientras que `'cat' in spam.values()` comprueba si hay un valor `'cat'` para una de las claves de `spam`.

7. `spam.setdefault('color', 'black')`.

8. `pprint.pprint()`.

## Capítulo 8

1. Los caracteres de escape representan caracteres en valores de cadena que de otro modo sería difícil o imposible poner en código.

2. El carácter de escape `\n` es un salto de línea y `\t` es un tabulador.

3. El carácter de escape `\\` representará una barra invertida.

4. La comilla simple de Howl's está bien porque hemos usado comillas dobles para marcar el principio y el final de la cadena.

5. Las cadenas multilínea permiten usar saltos de línea en cadenas sin el carácter de escape `\n`.



6.Las expresiones se evalúan así:

- ‘e’.
- ‘Hello’.
- ‘Hello’.
- ‘lo world!.

7.Las expresiones se evalúan así:

- ‘HELLO’.
- True.
- ‘hello’.

8.Las expresiones se evalúan así:

- [‘Remember,’ ‘remember,’ ‘the’ ‘fifth’ ‘of’ ‘November.’.
- ‘There-can-be-only-one.’.

9.Los métodos de cadena `rjust()`, `ljust()` y `center()`, respectivamente.

10.Los métodos `rstrip()` y `rstrip()` eliminan espacios en blanco desde los extremos izquierdo y derecho, respectivamente, de una cadena.

## Capítulo 9

1.La función `re.compile()` crea objetos `Regex`.

2.Las cadenas crudas se usan para no tener que usar caracteres de escape para las barras invertidas.

3.El método `search()` devuelve objetos `Match`.

4.El método `group()` devuelve cadenas del texto coincidente.

5.El grupo 0 es la coincidencia entera, el grupo 1 cubre el primer par de paréntesis y el grupo 2 el segundo juego de paréntesis.

6. Se puede usar el carácter de escape barra invertida con puntos y paréntesis: `\.`, `\(` y `\)`.
7. Si la expresión regular no tiene grupos, devuelve una lista de cadenas. Si tiene grupos, devuelve una lista de tuplas de cadenas.
8. La barra vertical (`|`) significa buscar “o, o” entre dos grupos.
9. El carácter `?` puede significar “busca cero o uno del grupo anterior” o usarse para indicar que una búsqueda no debe ser voraz.
10. `+` busca uno o más. `*` busca cero o más.
11. `{3}` busca exactamente tres apariciones del grupo precedente. `{3,5}` busca entre tres y cinco apariciones.
12. Las clases de caracteres estenográficas `\d`, `\w` y `\s` buscan un solo dígito, palabra o espacio, respectivamente.
13. Las clases de caracteres estenográficas `\D`, `\W` y `\S` buscan un solo carácter que no sea un dígito, una palabra o un espacio, respectivamente.
14. `*` realiza una búsqueda voraz y `.*?` hace una búsqueda no voraz.
15. `O` `[0-9a-z` o `[a-z0-9`.
16. Pasar `re.I` o `re.IGNORECASE` como segundo argumento a `re.compile()` hará que la búsqueda no distinga mayúsculas y minúsculas.
17. El carácter `.` suele buscar cualquier carácter, excepto el salto de línea. Si pasamos `re.DOTALL` como segundo argumento a `re.compile()`, buscará también los saltos de línea.
18. La llamada a `sub()` devolverá la cadena ‘X drummers, X pipers, five rings, X hens’.
19. El argumento `re.VERBOSE` permite añadir espacio en blanco y comentarios a la cadena pasada a `re.compile()`.

- 1.Las rutas relativas son relativas al directorio de trabajo actual.
- 2.Las rutas absolutas empiezan con el directorio raíz, como / o C:\.
- 3.En Windows, se evalúa a `WindowsPath('C:/Users/Al')`. En otros sistemas operativos, se evalúa a otro tipo de objeto `Path` pero con la misma ruta.
- 4.La expresión `'C:/Users' / 'Al'` da error, ya que no se puede usar el operador `/` para unir dos cadenas.
- 5.La función `os.getcwd()` devuelve el directorio de trabajo actual. La función `os.chdir()` cambia el directorio de trabajo actual.
- 6.La carpeta `.` es la carpeta actual y `..` la carpeta primaria.
- 7.`C:\bacon\eggs` es el directorio y `spam.txt` la base del nombre.
- 8.La cadena `'r'` para el modo lectura, `'w'` para el modo escritura y `'a'` para el modo agregar.
- 9.Un archivo existente que se abra en modo escritura se borra y sobrescribe por completo.
- 10.El método `read()` devuelve el contenido completo del archivo como un único valor de cadena. El método `readlines()` devuelve una lista de cadenas, donde cada cadena es una línea del contenido del archivo.
- 11.Un valor estante se parece a un valor de diccionario; tiene claves y valores, además de métodos `keys()` y `values()` que funcionan de manera similar a los métodos de diccionario homónimos.

## Capítulo 11

- 1.La función `shutil.copy()` copia un solo archivo, mientras que `shutil.copytree()` copia una carpeta entera, con todo su contenido.
- 2.La función `shutil.move()` sirve para renombrar archivos, además de para moverlos.
- 3.La función de `send2trash` moverá un archivo o carpeta a la

Papelera de reciclaje, mientras que la de `shutil` eliminará de manera permanente el archivo o carpeta.

4. La función `zipfile.ZipFile()` es el equivalente de `open()`; el primer argumento es el nombre de archivo y el segundo, el modo para abrir el archivo ZIP (lectura, escritura o agregar).

## Capítulo 12

1. El comando `dir` enumera el contenido de una carpeta en Windows. El comando `ls` hace lo mismo en macOS y Linux.

2. La variable de entorno `PATH` contiene una lista de carpetas que se comprueban cuando se introduce un nombre de programa en el terminal.

3. La variable `_file_` contiene el nombre de archivo del *script* de Python que está en ejecución. Esta variable no existe en el intérprete de comandos interactivo.

4. El comando `cls` limpia el terminal en Windows, mientras que `clear` lo hace en macOS y Linux.

5. Ejecuta `python -m venv.venv` en Windows, o `python3 -m venv.venv` en macOS y Linux.

6. Ejecuta `python -m PyInstaller --onefile yourScript.py` en Windows, o `python3 -m PyInstaller --onefile yourScript.py` en macOS y Linux.

## Capítulo 13

1. El módulo `webbrowser` tiene un método `open()` que abrirá un navegador web con una URL específica, sin más. El módulo `requests` puede descargar archivos y páginas de la red. El módulo `bs4` analiza HTML.

2. La función `requests.get()` devuelve un objeto `Response`, que tiene un atributo de texto que contiene el contenido descargado como una cadena.

3. El método `raise_for_status()` lanza una excepción si la descarga ha

tenido problemas y no hace nada si ha ido bien.

4.El atributo `status_code` del objeto `Response` contiene el estado del código HTTP.

5.Una vez abierto el nuevo archivo en el ordenador en modo `'wb'`, usa un bucle `for` que itera por el método `iter_content()` del objeto `Response` para escribir fragmentos en el archivo. Aquí tienes un ejemplo:

```
saveFile = open('filename.html', 'wb')
```

```
for chunk in res.iter_content(100000):
```

```
saveFile.write(chunk)
```

6.La mayoría de las API devuelven sus respuestas en formato JSON o XML.

7.**F12** muestra las herramientas para desarrolladores en Chrome. Para verlas en Firefox, pulsa **Control-Mayús-C** (en Windows y Linux) or **Comando-Opción-C** (en OS X).

8.Haz clic con el botón secundario del ratón en el elemento y selecciona Inspeccionar en el menú contextual.

9.`'#main'`.

10.`'highlight'`.

11.`spam.getText()`.

12.`linkElem.attrs`.

13.El módulo `selenium` se importa con `from selenium import webdriver`.

14.Los métodos del elemento `find_element_*` devuelven el primer elemento coincidente como un objeto `WebElement`. Los métodos `find_elements_*` devuelven una lista de todos los métodos coincidentes como objetos `WebElement`.

15. Los métodos `click()` y `send_keys()` simulan clics de ratón y pulsaciones de teclado, respectivamente.
16. El método `press('Control + A')` simula pulsar **Control-A**.
17. Los métodos `forward()`, `back()` y `refresh()` del objeto `WebDriver` simulan esos botones del navegador.
18. Los objetos `Page` `go_forward()`, `go_back()` y `reload()` simulan esos botones del navegador.

## Capítulo 14

1. La función `openpyxl.load_workbook()` devuelve un objeto `Workbook`.
2. El atributo `sheetnames` contiene una lista de cadenas con los títulos del libro.
3. Ejecutando `wb['Sheet1']`.
4. Utilizando `wb.active`.
5. Con `sheet['C5'].value` o `sheet.cell(row=5, column=3).value`.
6. Con `sheet['C5'] = 'Hello'` o `sheet.cell(row=5, column=3).value = 'Hello'`.
7. Con `cell.row` y `cell.column`.
8. Contienen la columna y la fila más altas con valores en la hoja, respectivamente, como valores enteros.
9. A `openpyxl.cell.column_index_from_string('M')`.
10. A `openpyxl.cell.get_column_letter(14)`.
11. Con `sheet['A1':'F1']`.
12. Con `wb.save('example3.xlsx')`.
13. Las fórmulas se establecen como cualquier otro valor. Hay que

establecer el atributo `value` de la celda como una cadena con el texto de la fórmula. Recuerda que las fórmulas empiezan con un igual (=).

14. Pasar `data_only = True` al llamar a `load_workbook()` para que OpenPyXL recupere el resultado de la fórmula calculado.

15. Con `sheet.row_dimensions[5].height = 100`.

16. Con `sheet.column_dimensions['C'].hidden = True`.

17. Los paneles inmovilizados son filas y columnas que siempre aparecen en la pantalla. Son útiles para encabezados.

18. `openpyxl.chart.Reference()`, `openpyxl.chart.Series()`, `openpyxl.chart.BarChart()`, `chartObj.append(seriesObj)` y `add_chart()`.

## Capítulo 15

1. Para acceder a Hojas de cálculo de Google, hace falta un archivo de credenciales, un archivo de *tokens* para Hojas de cálculo de Google y un archivo de *tokens* para Google Drive.

2. EZSheets tiene objetos `ezsheets.Spreadsheet` y `ezsheets.Sheet`.

3. Llamando al método de `Spreadsheet` `downloadAsExcel()`.

4. Llamando a la función `ezsheets.upload()` y pasándole el nombre del archivo de Excel.

5. El valor de `ss['Students']['B2']`.

6. Llamando a `ezsheets.getColumnLetterOf(999)`.

7. Accediendo a las propiedades `rowCount` y `columnCount` del objeto `Sheet`.

8. Llamando al método `delete()` de `Sheet`. La eliminación solo es permanente si pasamos el argumento `permanent = True`.

9. La función `Spreadsheet()` y el método `Sheet()` de `Spreadsheet`

crearán objetos Spreadsheet y Sheet, respectivamente.

10.EZSheets reduciría las llamadas a métodos.

## Capítulo 16

1.conn = sqlite3.connect('example.db', isolation\_level=None).

2.conn.execute('CREATE TABLE students (first\_name TEXT, last\_name TEXT, favorite\_color TEXT) STRICT').

3.Pasando el argumento isolation\_level=None al llamar a sqlite3.connect().

4.Mientras que INTEGER es análogo al tipo int de Python, REAL equivale al tipo float.

5.El modo estricto añade un requisito de que todas las columnas deben tener un tipo de datos, y SQLite lanza una excepción si intentas insertar datos del tipo incorrecto.

6.El asterisco (\*) significa “selecciona todas las columnas de la tabla”.

7.CRUD viene de *Create* (crear), *Read* (leer), *Update* (actualizar) y *Delete* (borrar), las cuatro operaciones estándar que hacen las bases de datos.

8.ACID viene de *Atomicity* (atomicidad), *Consistency* (consistencia), *Isolation* (aislamiento) y *Durability* (durabilidad), las cuatro propiedades que deberían tener las transacciones de una base de datos.

9.Las instrucciones INSERT añaden registros nuevos a una tabla.

10.Las instrucciones DELETE eliminan registros de una tabla.

11.Sin una proposición WHERE, la instrucción UPDATE se aplicará a todos los registros de la tabla, que puede ser o no ser lo que realmente queríamos hacer.

12.Un índice es una estructura de datos que organiza los datos de



una columna, lo cual ocupa un poco más de espacio, pero hace las consultas más deprisa. 'CREATE INDEX idx\_birthdate ON cats (birthdate)' crearía un índice para la columna birthdate de la tabla cats.

13.Una clave foránea vincula registros de una tabla con un registro de otra.

14.Podemos borrar una tabla llamada cats ejecutando la instrucción 'DROP TABLE cats'.

15.La cadena ':memory:' se usa en lugar de un nombre de archivo para crear una base de datos en memoria.

16.El método iterdump() puede crear las instrucciones para copiar una base de datos. También se puede copiar el propio archivo de la base de datos.

## Capítulo 17

1.Un objeto File debe abrirse en modo escritura pasando 'w' a open().

2.Llamar a getPage(4) devolverá un objeto Page para la página 5, ya que 0 es la primera página.

3.Llamar a decrypt('swordfish').

4.Para rotar páginas en sentido contrario al de las agujas del reloj, hay que pasar al mismo método enteros negativos: -90, -180 o -270.

5.docx.Document('demo.docx').

6.Los objetos Paragraph representan párrafos de texto y se componen de uno o varios objetos Run.

7.Usa doc.paragraphs para obtener una lista de objetos Paragraph.

8.El objeto Run tiene esas variables (el objeto Paragraph, no).

9.True siempre pone el objeto Run en negrita y False siempre hace que no esté en negrita, sea cual sea la configuración del estilo. None

hará que el objeto Run use solo la configuración en negrita del estilo.

10.Llamando a la función docx.Document().

11.Con doc.add\_paragraph('Hello there!').

12.Los enteros de 0 a 9.

## Capítulo 18

1.En Excel, las hojas de cálculo pueden tener valores de tipos de datos distintos de las cadenas; las celdas pueden tener distintas fuentes, tamaños y configuraciones de color; las celdas pueden tener distintas alturas y anchuras; las celdas adyacentes se pueden fusionar y podemos incrustar gráficos e imágenes.

2.Se pasa un objeto File obtenido de una llamada a open().

3.Los objetos File tienen que abrirse en modo lectura binaria ('rb') para los objetos Reader y escritura binaria ('wb') para los objetos Writer.

4.El método writerow().

5.El argumento delimiter cambia la cadena usada para separar las celdas de una fila. El argumento lineterminator cambia la cadena empleada para separar filas.

6.Todos ellos pueden editarse fácilmente con un editor de texto: CSV, JSON y XML son formatos de texto simple.

7.json.loads().

8.json.dumps().

9.El formato de XML se parece a HTML.

10.JSON representa los valores None con la palabra clave null.

11.Los valores booleanos se escriben en minúsculas en JSON: true y false.

## Capítulo 19

- 1.Un momento de referencia que emplean muchos programas de fecha y hora. Ese momento es el 1 de enero de 1970, UTC.
- 2.time.time().
- 3.time.asctime().
- 4.Con time.sleep(5).
- 5.Devuelve el entero más próximo al argumento pasado. Por ejemplo, round(2.4) devuelve 2.
- 6.Un objeto datetime representa un momento específico. Un objeto timedelta representa una duración.
- 7.Ejecuta datetime.datetime(2019, 1, 7).weekday(), que devuelve 0. Esto significa lunes, ya que el módulo datetime usa 0 para el lunes, 1 para el martes y así sucesivamente hasta 6 para el domingo.

## Capítulo 20

- 1.Los archivos credentials.json y token.json dicen al módulo EZGmail qué cuenta de Google usar para acceder a Gmail.
- 2.Un mensaje representa un solo correo, mientras que una conversación con varios correos es un hilo.
- 3.Incluye el texto 'has:attachment' en la cadena que pases a search().
- 4.Las pasarelas de correo electrónico a SMS no tienen garantías de funcionar, no notifican si se ha entregado el mensaje y el que hayan funcionado una vez no significa que vayan a funcionar dos.
- 5.La biblioteca Requests puede enviar y recibir notificaciones de ntfy.

## Capítulo 21

- 1.Un valor RGBA es una tupla de cuatro enteros, que van de 0 a 255. Estos enteros corresponden a la cantidad de rojo, verde, azul y alfa (transparencia) del color.
- 2.Llamar a `ImageColor.getcolor('CornflowerBlue', 'RGBA')` devolverá (100, 149, 237, 255), el valor RGBA de ese tono de azul.
- 3.Una tupla de cuadro es un valor de tupla de cuatro enteros: la coordenada x del borde izquierdo, la coordenada y del borde superior, la anchura y la altura, respectivamente.
- 4.`Image.open('zophie.png')`.
- 5.`im.size` es una tupla de dos enteros, la anchura y la altura.
- 6.`im.crop((0, 50, 50, 50))`. Observa que estás pasando una tupla de cuadro a `crop()`, no cuatro argumentos enteros separados.
- 7.Llamando al método `im.save('new_filename.png')` del objeto `Image`.
- 8.El módulo `ImageDraw` contiene código para dibujar sobre imágenes.
- 9.Los objetos `ImageDraw` tienen métodos de dibujo de formas, como `point()`, `line()` o `rectangle()`. Se devuelven pasando el objeto `Image` a la función `ImageDraw.Draw()`.
- 10.`plt.plot()` crea un gráfico de líneas, `plt.scatter()` crea un diagrama de dispersión, `plt.bar()` crea un gráfico de barras, y `plt.pie()` crea un gráfico circular.
- 11.El método `savefig()` guarda el gráfico como una imagen.
- 12.No se puede llamar a `plt.show()` dos veces seguidas porque reestablece los datos del gráfico, lo que nos obliga a ejecutar de nuevo el código que crea el gráfico si queremos mostrarlo por segunda vez.

- 1.Tesseract reconoce el inglés por defecto.

2. PyTesseract funciona con la biblioteca de imágenes Pillow.
3. La función `image_to_string()` acepta un objeto `Image` y devuelve una cadena.
4. No, Tesseract solo extrae texto de documentos escaneados de texto mecanografiado, no de fotos.
5. `tess.get_languages()` devuelve una lista de cadenas de los paquetes de idiomas.
6. Podemos pasar el argumento `lang = 'eng + jpn'` para identificar el texto en inglés y en japonés en una imagen.
7. NAPS2 se puede ejecutar desde un *script* de Python para crear archivos PDF con texto de OCR incrustado.

## Capítulo 23

1. Moviendo el ratón a cualquier esquina de la pantalla.
2. La función `pyautogui.size()` devuelve una tupla con dos enteros para la anchura y la altura de la pantalla.
3. La función `pyautogui.position()` devuelve una tupla con dos enteros para las coordenadas *x* e *y* del cursor del ratón.
4. La función `moveTo()` mueve el ratón a coordenadas absolutas en la pantalla, mientras que `move()` mueve el ratón en relación a su posición actual.
5. `pyautogui.dragTo()` y `pyautogui.drag()`.
6. `pyautogui.typewrite('Hello world!')`.
7. Pasando una lista de cadenas de teclas a `pyautogui.write()` (como 'left') o pasando una sola cadena de tecla a `pyautogui.press()`.
8. Con `pyautogui.screenshot('screenshot.png')`.
9. `pyautogui.PAUSE = 2`.

10. Deberíamos usar Selenium para controlar el navegador en vez de PyAutoGUI.

11. PyAutoGUI hace clic y escribe a ciegas y no es fácil saber si está haciendo clic o escribiendo en la ventana correcta. Una ventana que emerja inesperadamente o un error puede hacer que el *script* se desvíe y que tengamos que cerrarlo.

12. Llamando a la función `pyautogui.getWindowsWithTitle('Notepad')`.

13. Ejecutando `w = pyatougui.getWindowsWithTitle('Firefox')` y, a continuación, `w.activate()`.

## Capítulo 24

1. Llamando a `engine.setProperty('rate', 300)`, por ejemplo, para hacer que la voz de `pyttsx3` diga 300 palabras por minuto.

2. El módulo `pyttsx3` guarda el audio en formato WAV.

3. No, `pyttsx3` y Whisper no requieren un servicio en línea ni acceso a Internet.

4. Sí, `pyttsx3` y Whisper son compatibles con lenguas distintas del inglés.

5. El modelo predeterminado de Whisper es 'base'.

6. SRT (*SubRip Subtitle*) y VTT Web (*Video Text Tracks*) son dos formatos habituales de archivos de subtítulos.

7. Sí, `yt-dlp` puede descargar de cientos de sitios web de vídeo, además de YouTube.

Título de la obra original:  
*Automate the Boring Stuff with Python: Practical Programming for Total Beginners,*  
*3rd Edition*

Edición en formato digital: febrero de 2026

Copyright © 2025 by Al Sweigart.

Title of English-language original: *Automate the Boring Stuff with Python: Practical Programming for Total Beginners, 3rd Edition* ISBN 978-1-7185-0340-3, published by No Starch Press Inc. 245 8th Street, San Francisco, California United States 94103. The Spanish-language 1st edition Copyright © 2026 by Grupo Anaya, S. A. U. (Anaya Multimedia) under license by No Starch Press Inc. All rights reserved.

© de la traducción: María Pascual Cabrerizo

© EDICIONES ANAYA MULTIMEDIA (GRUPO ANAYA, S. A.), 2026  
Calle Valentín Beato, 21  
28037 Madrid

ISBN ebook: 978-84-415-5292-0

Está prohibida la reproducción total o parcial de este libro electrónico, su transmisión, su descarga, su descompilación, su tratamiento informático, su almacenamiento o introducción en cualquier sistema de repositorio y recuperación, en cualquier forma o por cualquier medio, ya sea electrónico, mecánico, conocido o por inventar, sin el permiso expreso escrito de los titulares del Copyright.